NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

AN UPGRADEABLE AGENT-BASED MODEL TO EXPLORE NON-LINEARITY AND INTANGIBLES IN PEACEKEEPING OPERATIONS

by

Wolfgang Lehmann

June 2006

Thesis Advisor: Thomas W. Lucas
Second Reader: Arnold H. Buss

Approved for public release; distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** June 2006	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: An Upgradeable Agent-Based Model to Explore Non-Linearity and Intangibles in Peacekeeping Operations	**5. FUNDING NUMBERS** N/A
6. AUTHOR(S) Wolfgang Lehmann	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10.SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Peacekeeping operations (PKO) have become a significant challenge to the German Armed Forces. For the development of tactics, techniques, procedures and equipment with combat operations, agent-based models have been developed, used and exploited for many years. Modeling and simulation of PKO, however, is still in a very early stage. This thesis develops an agent-based model to analyze PKO. Unlike many other multi-agent systems (MAS), it implements the rules of discrete event simulation. The chosen software architecture makes the model upgradeable and useful for a breadth of future applications. The model's open architecture and the underlying principle of loosely coupled components make it easy to change or enhance the model. The software agents' design incorporates individuality, which is characterized by personality factors. Furthermore, the model is data-farmable. Required data inputs into the simulation tool, i.e., PKO scenarios, are formatted utilizing a state-of-the-art technology called Extensible Markup Language (XML), which facilitates use of the data in nearly all computer software packages. The model executes multiple runs of multiple scenarios automatically, demonstrating a robust nature. Finally, an exemplary analysis demonstrates data-farming concepts on the effect of personality factor settings on the potential escalation of a PKO scenario.

14. SUBJECT TERMS Modeling, Simulation, Peacekeeping Operations, Multi-Agent System, Agent-Based Simulation, Discrete Event Simulation, Data Farming, Design of Experiment	**15. NUMBER OF PAGES** 135
	16. PRICE CODE N/A

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

AN UPGRADEABLE AGENT-BASED MODEL TO EXPLORE NON-LINEARITY AND INTANGIBLES IN PEACEKEEPING OPERATIONS

Wolfgang Lehmann
Major, German Army
Dipl.-Ing.(FH), Universität der Bundeswehr München, 1994

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
June 2006

Author: Wolfgang Lehmann

Approved by: Thomas W. Lucas
Thesis Advisor

Arnold H. Buss
Second Reader

James N. Eagle
Chairman, Department of Operations Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Peacekeeping operations (PKO) have become a significant challenge to the German Armed Forces. For the development of tactics, techniques, procedures and equipment with combat operations, agent-based models have been developed, used and exploited for many years. Modeling and simulation of PKO, however, is still in a very early stage. This thesis develops an agent-based model to analyze PKO. Unlike many other multi-agent systems (MAS), it implements the rules of discrete event simulation. The chosen software architecture makes the model upgradeable and useful for a breadth of future applications. The model's open architecture and the underlying principle of loosely coupled components make it easy to change or enhance the model. The software agents' design incorporates individuality, which is characterized by personality factors. Furthermore, the model is data-farmable. Required data inputs into the simulation tool, i.e., PKO scenarios, are formatted utilizing a state-of-the-art technology called Extensible Markup Language (XML), which facilitates use of the data in nearly all computer software packages. The model executes multiple runs of multiple scenarios automatically, demonstrating a robust nature. Finally, an exemplary analysis demonstrates data-farming concepts on the effect of personality factor settings on the potential escalation of a PKO scenario.

THIS PAGE INTENTIONALLY LEFT BLANK

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs and data herein are free of computational, logic, and collection errors, they cannot be considered validated. Any application of these programs or data without additional verification is at the risk of the user.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ABBREVIATIONS, ACRONYMS, AND SYMBOLS

ABM	Agent-Based Model
ABS	Agent-Based Simulation
API	Application Programming Interface
aTa	AffinityToAction (variable name)
CAS	Complex Adaptive System
CD&E	Concept Development and Experimentation
CNA	Center for Naval Analyses
cTf	CloseToFriendly (variable name)
cTi	CloseToLeader (variable name)
DES	Discrete Event Simulation
EADS	European Aeronautic Defence and Space Corporation
FER	Force Exchange Ratio
fK	timeToFirstKill (variable name)
GUI	Graphical User Interface
IDE	Integrated Development Environment
IFOR	Implementation Force
ISAF	International Security Assistance Force
ISSAC	Irreducible Semi-Autonomous Adaptive Combat
JDOM	No acronym (Java Software package to process XML files)
KFOR	Kosovo Force
MANA	Map Aware Non-uniform Automata
MAS	Multi-Agent Systems
MHPCC	Maui High Performance Computing Center
MOE	Measure of Effectiveness
MOOTW	Military Operations Other Than War
MVCP	Model-View-Controller Paradigm
M&S	Modeling and Simulation
NPS	Naval Postgraduate School
oO	ObeyOrders (variable name)
OSCE	Organization for Security and Co-operation in Europe

OOTW	Operations Other Than War
PAX	No acronym (Latin word for peace)
PKO	Peacekeeping Operations
rA	RiskAversion (variable name)
SFOR	Stabilisation Force
sI	ShockInfluence (variable name)
Simkit	No acronym (Software package for DES)
SIRA	Simulatiuonsgestützte Rahmenübung
T	Training (variable name)
TSS	Time Step Simulation
UN	United Nations
UNOSOM	United Nations Operation in Somalia
UNPREDEP	United Nations Preventive Deployment
XML	eXtensible Markup Language

ACKNOWLEDGEMENTS

The author cannot believe that this project has finally come to a good end. Without the support of so many good people, this would never have ended successfully. Out of a great multitude of teachers, advisors, and friends, the following deserve special mention for the contributions they have made:

To Dr. Tom Lucas. Thank you for your encouragement, guidance, and support over such a long time. You have gone above the call of duty to make this project a success. It is such an honor for me having you as my thesis advisor.

To Dr. Arnold Buss. Thank you for developing Simkit, which is such a powerful simulation tool. Hopefully, my work pays back a little for your guidance on creating a software architecture that is flexible and open to future upgrades.

To Professor John Hiles. Thank you for catching my increasing interest in the field of multi-agent systems.

To Dr. Paul and Dr. Susan Sanchez. Thank you for opening my eyes to what a simulation model can tell me. I'll never forget our inspiring discussions.

To Captain Michael Margolis and Lieutenant Colonel Dr. Wellbrink. Words cannot express the whole extent of your support, encouragement, and inspiration. Thank you very, very much. I am truly blessed to call you my friends.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Since 2003, the German Federal Armed Forces (Bundeswehr) have been in a state of continuous transformation. The purpose of this transformation is to increase the operational and mission capabilities of the Bundeswehr force. In order to reach this goal, the Bundeswehr's missions, capabilities, and equipment need to be synchronized with the available budget. Therefore, the Bundeswehr missions were updated for military operations other than war (MOOTW), such as peacekeeping operations (PKO) and peace support—including the war against terror. These updated mission types now serve as the basis for current Bundeswehr doctrine. The Bundeswehr is undergoing a continuous process of self-evaluation, and thus continuous transformation, where future capabilities and force structure are determined by the ever-changing "most likely" mission set.

While the methodology of Concept Development and Experimentation (CD&E) is critical to the transformation process, Modeling and Simulation (M&S) also serves as an important analytical methodology for the Bundeswehr. M&S supports CD&E methodology in four major areas:

1) Analyzing missions and field exercises.
2) Evaluating courses of actions (wargaming).
3) Testing innovative concepts.
4) Supporting a multitude of analyses (e.g., analysis of capabilities, fulfillment of demand, and procurement).

The Bundeswehr Concept points out that the use of M&S supports process evaluation, validation, and verification. Through an iterative approach, Bundeswehr efficiency is increased. Therefore, M&S is an important tool for Bundeswehr transformation.

PKO M&S is in its infancy. Since classical combat models do not facilitate PKO M&S, the development of new tools is necessary. Multi-Agent Systems (MAS) are simulation models that consist of software entities called "agents." In information systems and artificial intelligence, software agents are widely understood to be representations of decision-making entities. There are a variety of software agents, of

which the class of reactive agents was chosen for the modeling of PKO. For such agents modeling a PKO scenario, previous work has demonstrated a distinction between outer and inner environments to be a useful principle. In this context, the outer environment would be the representation of a real-world, peacekeeping scenario. On the other hand, the inner environment only exists inside an agent. The inner environment is the representation of the agent's perceptions. The basic principle of MAS is based on the fact that even if individual strategies of single agents are simple, through independent activities the behavior of the system as a whole may become complex. The actions of individual agents, and the interactions between multiple agents, are based on factors such as their perceptions, inscribed rules, goals, and intentions.

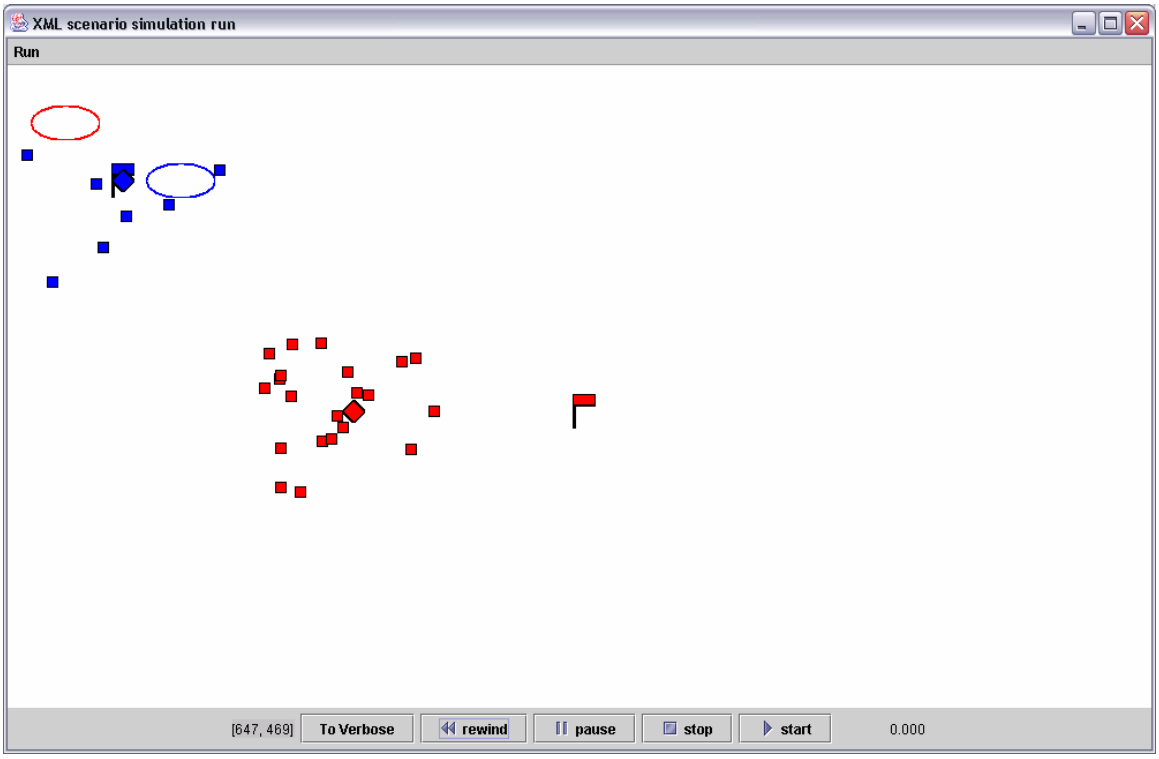
Building upon previous work, this thesis develops a MAS tool with the following key features:

1) It is based on a discrete event simulation (DES) approach.
2) It is based on a simulation package for DES called Simkit.
3) It has a flexible, open architecture that facilitates further upgrades.
4) It follows the model-view-controller paradigm.
5) It facilitates data-farming techniques.

The flexible building block structure of Java interfaces was chosen for the architecture of this tool due to Java's ability to facilitate upgradeability. The tool's architecture facilitates enhancement of single elements without having to change the other elements. For example, *Agent* (simulation entities are displayed in *italics*) is a Java interface that provides a frame for various implementations. The implementation provided in this model is conducted in the *BasicAgent* class. Future implementations of the *Agent* interface may be incorporated in the model without making fundamental changes throughout the entire program.

The model takes advantage of an animation package provided by Simkit. The following figure depicts a scenario in a two-dimensional (2-D) representation. The small squares represent humans—the blue ones peacekeepers, the red ones demonstrators. Diamonds represent leaders. Flags represent the base of each side and the ellipses are the objectives. As the reader can observe, the group of peacekeepers are at their objective,

whereas the demonstrators need to pass the peacekeepers on their way towards the red objective. As soon as the peacekeepers sense demonstrators (and vice versa), the events begin to escalate—to include possible weapon discharges in the air and discharges aimed at person entities.

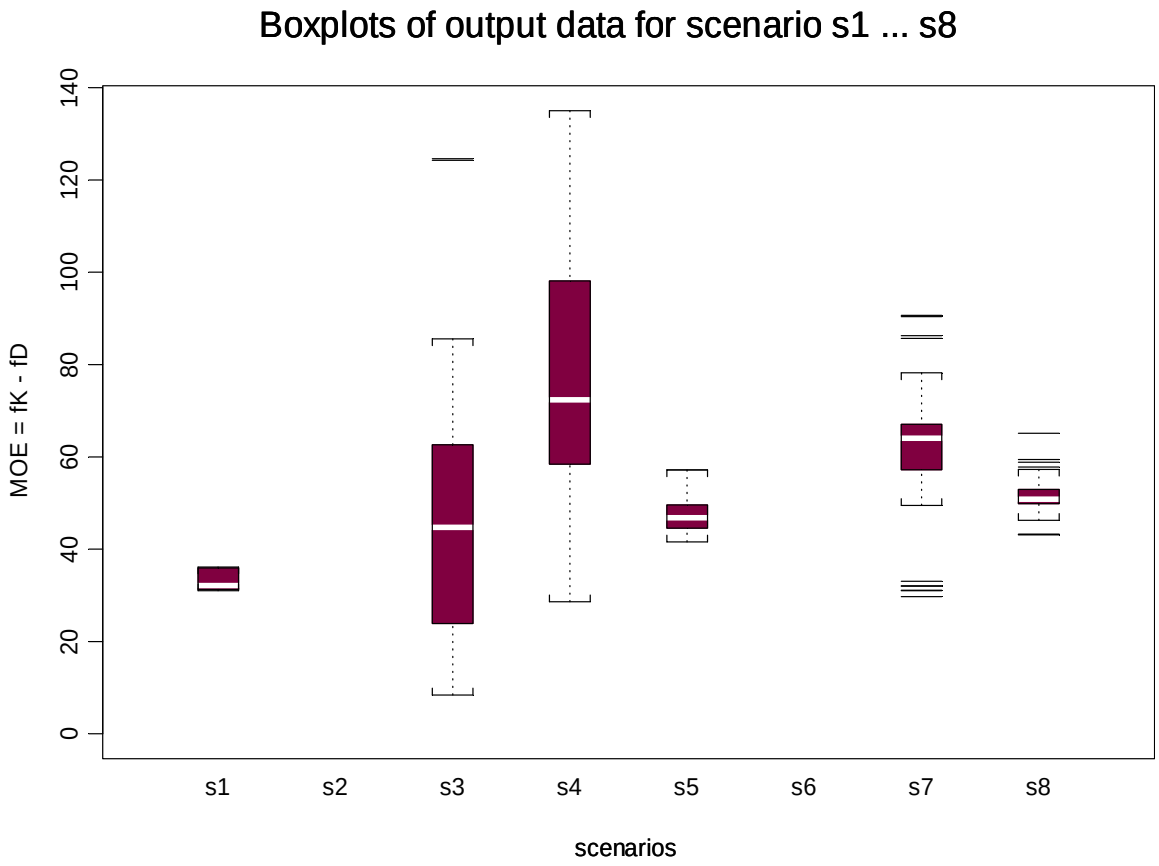


Scenario for simulation in a 2-D animation

The agents' individuality is characterized by seven personality factors: CloseToFriendly (cTf), ObeyOrders (oO), CloseToLeader (cTl), AffinityToAction (aTa), RiskAversion (rA), ShockIfluence (sI), and Training (t). In an exemplary analysis, this thesis demonstrates how the model facilitates data-farming techniques. Out of these seven personality factors, three (cTf, aTa, and rA) were picked to apply a two-level, full factorial design. Each of the resultant eight scenarios was replicated 100 times, for a total of 800 computational experiments.

To sum up the overall results of the exemplary analysis, the combination of a low aTa and a high rA is very unfortunate for our analysis since the situation does not escalate (scenarios s2 and s6)—for PKOs a highly desirable outcome. However, analysis of single runs with such settings show that the red agents finally reach their objective;

i.e., pass the group of blue agents whose duty it is to keep them from getting there. The possibility of using force needs to be considered with the ability and the attitude to do so, if necessary. The results of other combinations of input variables are shown in the following boxplot diagram. The measure of effectiveness (MOE) for the eight scenarios (i.e., the y-axis) is the difference between the time of the first detection and the time of the first kill.



Boxplots graphs of the output data

In order to study different combinations of input factors, this thesis uses two common data-farming techniques: a classification tree and a linear regression model.

Classification tree model: Applying the classification tree technique enables answering the following question: Which one of the three input factors has the greatest effect on the occurrence of kills? As it turns out, it is aTa. The second most influential variable is rA, followed by cTf. Furthermore, the tree model confirms the conclusion that the combination of aTa low and rA high results in no kills. On the other hand, if rA is

low and aTa is high, the model always suggests a kill, with a misclassification rate of 12%, no matter the setting of cTf. The attitude of soldiers represented by this input setting is definitely unfortunate for peacekeeping missions. The group of peacekeepers may be successful in keeping the crowd of demonstrators from reaching their objective; however, only at the price of people being killed.

Linear regression model: An MOE was defined in this analysis as the time to the first kill subtracted by the time to the first detection. In order to build a linear regression model, only those data points that included at least one kill can be used. The question here is: "Given a kill, what is the influence of the input factors on the MOE?" For this subset of complete output data, the linear regression model suggests that rA is the most influential input factor, followed by cTf and aTa. Also, there is a significant interaction between aTa and rA. All factors (except for the interaction term) are positive, thereby indicating that increasing each one results in a greater MOE. In other words, the model suggests that, given a kill, increasing the aTa of all agents causes the situation to escalate slower. This is a surprise. There is no intuitive explanation to this finding. Further research needs to be undertaken in order to gain more insight into this phenomenon. Such surprises that cannot be explained immediately are not new to the world of agent-based simulation (ABS). In fact, the purpose of ABS is often to generate questions rather than answers. Through discussion of these questions with modelers, analysts, and other experts, decision makers may glean more insightful knowledge on the subject matter.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PURPOSE

Since 2003, the German Federal Armed Forces (Bundeswehr) has been in a process of continuous transformation. The goal of this transformation process is to increase the mission capability of the Bundeswehr force. In order to reach this goal, the missions, capabilities, and equipment of the forces need to be synchronized with the available budget. Therefore, the Bundeswehr's missions were reprioritized and military operations other than war (MOOTW), such as peacekeeping, peace support, etc.—including the war on terror—were determined to be its new main focus. Since the budget is limited, anything else is of secondary importance. Consequently, the Bundeswehr is undergoing a process of constant transformation where future capabilities and the force structure are determined by the "most likely" missions.

In this respect, the major challenges for the Bundeswehr currently are:

1) Adjusting the course of the forces as a whole to the reprioritized set of missions in order to make them fit for combined operations.
2) Improving the joint forces' capabilities rather than the capabilities of the services—that is, applying a joint mindset and joint actions.
3) Restructuring the forces as a whole into three categories: intervention forces, stabilization forces, and support forces.
4) Realizing network centric warfare capabilities in order to become fully interoperable in multinational coalitions.
5) Adapting a new mission approach, including the presence of specific capabilities for limited durations in dynamic situations.
6) Changing the force structure so that formations can be assigned to missions as they are.
7) To undertake this transformation process while continuing to support missions worldwide.

Apart from the conceptual objectives, the Bundeswehr is already highly engaged in peace support and peacekeeping operations (PKO). At the present time, it is involved in missions on three continents with a total number of approximately 6,500 personnel. This number has been slowly and consistently increasing over the past ten years.

Consequently, the German Defence Policy Guidelines (Verteidigungspolitische Richtlinien) mention first among other Bundeswehr missions that:

> International conflict prevention and crisis management—including the war against terror—are the most likely missions of the German forces and a challenge of particular importance. These missions determine capabilities, command and control, availability and equipment of the Bundeswehr. These missions are not very different from missions in support of allies and may develop into such. Both types of missions require the same range of military capabilities. [Federal Minister of Defence, 2003]

In this environment of continuous transformation, Modeling and Simulation (M&S) has developed into a very important field to the Bundeswehr. The Bundeswehr Concept (Konzeption der Bundeswehr) states:

> Concept Development and Experimentation is a substantial methodology to the implementation of the transformation process. This methodology allows the identification of innovation potential, the evaluation of its relevance for the Bundeswehr, the examination of its reliability, the exploration of its effectiveness and thus the development of a solution for future concepts, methods, structures and/or systems. […] Concept Development and Experimentation supports the transformation of the Bundeswehr taking advantage of modeling and simulation by:
>
> 1) Analyzing missions and field exercises.
>
> 2) Evaluating courses of actions (wargaming).
>
> 3) Testing innovative concepts.
>
> 4) Supporting all kinds of analysis, e.g., analysis of capabilities, fulfillment of demand and procurement.
>
> The consequent application of Modeling and Simulation allows an evaluation of various processes, to redesign them in order to make them more efficient and thus can be a support to the whole transformation process. [Federal Minister of Defence, 2004]

The modeling of PKO, however, is still in its very early stages in the Bundeswehr and around the world. Previous approaches to model peacekeeping scenarios with modified combat models showed that there are very rigid bounds. One example is in the use of measures of effectiveness (MOEs). In many combat models the number of killed agents in any given fighting force, or the force exchange ratio (FER), determines the

result(s) of a simulation run. Consequently, the underlying goal on both sides is to cause as many casualties on the enemies' side as possible. This approach is very unfortunate for PKO. In peacekeeping scenarios, usually at least one side tries to avoid any casualties. Therefore, an analyst who wants to use a combat model to conduct analysis on peacekeeping is very limited. Many scenarios may even be impossible to be simulated. In the scenarios that can be simulated, the output may not contain any measures of interest. Therefore, there is a real need for models that are designed from the beginning to model MOOTW.

"Agent based simulation of German peacekeeping operations for units up to platoon level" was a first step. Erlenbruch [2002] executed an agent-based model (ABM) of PKO, thereby showing that ABMs are capable of simulating peacekeeping scenarios. Unfortunately, the current version of Erlenbruch's model does not contain features that allow for the creation of large quantities of data to be analyzed and synthesized. In order to thoroughly analyze a given scenario, the model needs to be uncoupled from the graphical user interface (GUI) and the output data needs to be data-farmable. "Data Farming is the process of using a high performance computer or computing grid to run a simulation thousands or millions of times across a large parameter and value space." [Wikipedia, http://en.wikipedia.org/wiki/Data_farming, June 2006] A software design that allows further enhancements is also desirable. Since the German Army is interested in the simulation of PKO, the logical next step is to conduct a redesign of Erlenbruch's model to facilitate data farming and make the model extensible.

B. BACKGROUND

1. Bundeswehr in Peacekeeping Operations (PKO)

In 1990, after a short period of international negotiations, Germany was reunified. For Germany, the results of World War II were finally overcome and the country regained its full sovereignty. For the Bundeswehr, it was the beginning of a period of change that continues to this day. The former East German forces, the National People's

Army (Nationale Volksarmee), had to be integrated with the Bundeswehr, while the *total* number of forces, both personnel and material, had to be reduced drastically according to the 2 + 4 Treaty.[1]

The German government soon learned that full sovereignty is accompanied by increased responsibility. The German armed forces—initially introduced to defend the country—were now ordered to participate in United Nations (UN) missions. Although the Bundeswehr was struggling with lower budgets, changes in force structures, and force reductions, the German government nevertheless eventually started sending troops to various places around the world. In 1992, for example, the German Army's Medical Corps established a field hospital in Cambodia. Only two years later, 1,700 German soldiers were stationed in Belet Uen, Somalia, to participate in the UNOSOM[2] mission. In the aftermath of the Gulf War, Germany also provided helicopters to support the UN Special Commission for the identification and destruction of chemical, biological, and nuclear weapons in Iraq. Currently, the Organization for Security and Co-operation in Europe (OSCE) mission to monitor the cease-fire in Georgia is supported by Bundeswehr medical personnel and observers. Since December 1996, Germany has sent more soldiers on the various UN missions in the former Yugoslavia, such as IFOR,[3] SFOR[4] (both in Bosnia-Herzegovina), UNPREDEP[5] (Macedonia), and KFOR[6] (Kosovo), than any other European country. On February 10, 2003, the 1st German-Dutch Corps took over ISAF[7] command and succeeded the Turkish Army as lead nation to the UN mission in Afghanistan. [http://www.einsatz.bundeswehr.de/C1256F1D0022A5C2/CurrentBaseLink/W2698QP3402INFODE#headerblock, June 2006]

---

[1] The "2+4 Treaty" of 12 September 1990 is the document by which the four former occupying powers, USA, Great Britain, France, and the Soviet Union, ceased to exert their rights in Germany.

[2] United Nations Operation in Somalia.

[3] Implementation Force (NATO-led multinational force in Bosnia and Herzegovina, 1995 - 1996)

[4] Stabilization Force (NATO-led multinational force in Bosnia and Herzegovina, 1996 - 1998)

[5] United Nations Preventive Deployment.

[6] Kosovo Force.

[7] International Security Assistance Force.

Today, Germany has the second largest number of troops engaged in various UN PKO throughout the world. German forces have been changing continuously since the beginning of the Cold War, and its end forced the Bundeswehr to prepare nearly overnight for missions other than the defense of Germany proper.

For the Bundeswehr, PKO was the catalyst for these reforms. Procurement of new weapon systems and new, compatible methods of communication were necessary. However, the first change was an effort to revise the mind-set of personnel. New training concepts had to be developed to adequately prepare officers and enlisted personnel for their tasks in PKO.

2. Complex Adaptive Systems (CAS)

Previous work has proven that it is legitimate to think of peacekeeping scenarios as CAS. In computer simulations CAS are usually modeled as multi-agent systems (MAS). For a thorough study of CAS and MAS the reader may find useful recommendations in the List of References. Wellbrink (2003) in his dissertation and, in particular, Erlenbruch (2002) in his thesis summarize a set of definitions and derivations on the interdependence of CAS and MAS. To make it easier for the reader to get an idea, a short summary of definitions is provided in this work.

Wellbrink points out that intense research of CAS at the Santa Fe Institute goes back to the mid-1990s. He named John Holland as the father of genetic systems and provides Waldrop's ten most important points of Holland's lecture:

1) First each of these systems is a *network of many agents* acting in parallel.

2) Furthermore, the *control of a complex adaptive system is highly dispersed*. There is no master neuron in the brain, for example, nor is there any master cell within a developing embryo. If there is to be any coherent behavior in the system it has to arise from competition and cooperation among the agents themselves.

3) Second, a complex adaptive system has *many levels of organization, with agents at any one level serving as building blocks for agents at a higher level.* A group of proteins, lipids, and nucleic acids will form a cell; a group of cells will form a tissue; a collection of tissues will form an organ; etc.

4) Furthermore, said Holland—and this is something he considered very important—*complex adaptive systems are constantly revising and rearranging their building blocks as they gain experience*. Succeeding generations of organisms will modify and rearrange their tissues through the process of evolution. The brain will continually strengthen and weaken myriad connections between its neurons as an individual learns from his or her encounters with the world.

5) At some deep, fundamental level, all these processes of learning, evolution and adaptation are the same. And one of the fundamental mechanisms of adaptation in any given system is this revision and recombination of the building blocks.

6) Third, he said, all *complex adaptive systems anticipate the future*.

7) More generally, *every complex adaptive system is constantly making predictions based on its various internal models of the world - its implicit or explicit assumptions about the way things are out there*. Furthermore, these models are much more than passive blueprints. They are active. Like subroutines in a computer program, they can come to life in a given situation and 'execute,' producing behavior in the system. In fact, you can think of internal models as the building blocks of behavior. And like any other building blocks, they can be tested, refined, and rearranged as the system gets experience.

8) Finally, said Holland, complex adaptive systems typically have many *niches*, each one of which can be exploited by an agent adapted to fill that niche.

9) And that, in turn, means that it is essentially meaningless to talk about a complex adaptive system as in a state of equilibrium: the system can never get there. It is always unfolding, always in transition. In fact if the system ever does reach equilibrium, it isn't just stable. It's dead!

10) And by the same token, there's no point imagining the agents in the system can 'optimize' their fitness, or their utility, or whatever. The space of possibilities is too vast; they have no practical way of finding the optimum. The most they can ever do is change to improve themselves relative to what the other agents are doing. In short, complex adaptive systems are characterized by continuous novelty.

[Waldrop, 1992, p. 145]

In order to prove that there is no such thing as a consistent definition, Wellbrink provides Nobel Price winner Gell-Mann's explanation of CAS:

> A complex adaptive system acquires information about its environment and its own interaction with that environment and its own interaction with that environment, identifying regularities in that information, condensing those regularities into a kind of "schema" or model, and acting in the real world on the basis of that schema. In each case, there are various competing schemata, and the results of the action in the real world feed back to the influence the competition among those schemata. [Gell-Mann, 1994]

Wellbrink concludes that some researchers call the CAS approach the third way of doing research. Though providing insight to a problem domain, CAS do not function sufficiently as forecasting tools. Moreover, they are useful tools that show possible interactions and may produce emergent behavior that could potentially occur at some point.

Erlenbruch brought PKO and CAS into a close relationship. Moreover, he built on the hypothesis that a PKO actually is best modeled by a CAS. In order to justify this claim he applied the above definitions as follows:

> Complex adaptive systems are described in detail because the author believes that peacekeeping operations can be modeled as CAS. PKOs are dynamic systems composed of many nonlinearly-interacting parts. Entities in PKOs can be aggregated to soldiers, commanders, demonstrating civilians, and fearful children for example. Tagging takes place in PKOs, a number of soldiers belong to a platoon commanded by a platoon leader, and demonstrators build a group with a common goal. The interacting groups are composed of a number of nonlinearly interacting parts; sources include feedback loops in command and control hierarchy, interpretation of opponent actions, adaptation to opponent actions, decision-making process, and elements of chance. There are flows between the individuals in PKOs; these flows are mostly information. The individuals in these operations are also diverse; on both sides there are leaders and followers, heroes and individuals driven by fear. All participants in a PKO have their own internal model, which is created by their environment, and by the way they realize it. And finally the individuals use building blocks to represent their view of the surrounding environment. The building blocks for the soldiers depend on their training experiences and their orders; those of the civilians depend on their ideology, goals, and information. In conclusion: Peacekeeping operations possess all features of complex adaptive systems. Forces and groups are composed of a number of nonlinearly interacting parts and at least the forces are organized in a command and control hierarchy; local action, which often appears disordered, induces long-range order; groups, in order to fulfill their goals, must continually adapt to a changing environment.

> There is no master "voice" that dictates the actions of each and every entity; and so on. [Erlenbruch, 2002]

Given these illustrations the reader may get an idea why it is worth to make another attempt to model peacekeeping scenarios as a CAS.

3. Multi-Agent Systems (MAS)

Among the modeling and simulation community it is widely understood that CAS can be designed as MAS. As a matter of fact, one hardly finds a definition for both terms that clearly distinguishes between each other. In this work, we understand an MAS to be a computational tool designed to model some sort of a CAS.

In information systems and artificial intelligence software, agents are widely understood as representations of decision-making entities. Agents are created to perform tasks and they feature some combination of the following selected properties:

1) autonomy
2) activity
3) communicability
4) adaptability
5) mobility

Agents' properties are defined in terms of their tasks and their environment, which influences them and at the same time is affected by their behavior. Depending on the combination of tasks and the environment, the literature distinguishes between various categories of agents. Based on Wooldridge and Weiss, Erlenbruch named three of these categories:

> 1) Reactivity: intelligent agents are able to perceive their environment, and respond in a timely fashion to changes that occur in it in order to satisfy their design objectives;
>
> 2) Pro-activeness: intelligent agents are able to exhibit goal-directed behavior by taking the initiative in order to satisfy their design objectives;
>
> 3) Social ability: intelligent agents are capable of interacting with other agents in order to satisfy their design objectives. [Erlenbruch, 2002]

Now, what makes an MAS is the fact that even if the individual strategies of the single agents are simple, through their independent activities the behavior of the whole system may become very complex. Therefore, any kind of outside control must be eliminated. The interaction of agents is solely based on their inscribed rules, goals, intentions, and the perception of their environment. Usually, the outcome of an MAS is highly nonlinear. Because of the degree of the nonlinearities, intangibles, and unexpected outcomes, surprise has become the main focus of those analysts who use ABMs in military simulations. As a matter of fact, ABMs have been designed with their programmers expecting them to generate some output that would surprise the analyst. In an approach of backward-thinking "the analyst envisions an outcome and traces how this outcome might have become possible." [Wellbrink, 2003]

We leave this brief introduction on MAS with a quote of John Holland's:

> I just love these things where the situation unfolds and I say, 'Gee whiz! Did that really come from these assumptions!?' Because if I do it right, if the underlying rules of evolution of the themes are in control and not me, then I'll be surprised. And if I'm not surprised, then I am not very happy, because I know I've built everything in from the start. [Holland in Waldrop, 1992]

4. Modeling Peacekeeping Operations (PKO)

Although traditional military exercises gave way to exercises in MOOTW, M&S in the Bundeswehr remained mostly unaffected until 2000. For example, SIRA,[8] a tool for advanced tactical training of a battalion staff in traditional combined-weapons missions, was recently enhanced in order to simulate peace support operations. The Officer Training Center of the German Army provides a detailed overview of the system and its use. [http://www.offizierschule.de/hptzh/sira/, March 2003]. A wider overview on M&S in the German Army is given by [Stolte, 2001].

Although a variety of simulations for different combat missions are available, the simulation of PKO is still in its infancy. Project Albert is a good example. "The Marine Corps Warfighting Lab's Project Albert is the research effort to assess the general applicability of the study of complex adaptive systems to warfare, and to provide new

[8] Simulation Based Field Exercise (Simulationsgestuetzte Rahmenuebung).

methodologies for investigating the results of running such models, and incorporating those results with other, more traditional, methods of analysis." [http://www.projectalbert.org, March 2003] In annual workshops, military simulation analysts from various countries use relatively simple ABMs to address many post-Cold War issues. Project Albert working groups have successfully managed to model specific peacekeeping scenarios and were able to answer operational questions using modified models that were originally designed to model combat operations. [Horne, 2002]

Participating in Project Albert, the Bundeswehr funded the development of PAX.[9] PAX is a model that was designed to explore the development and escalation of violence in a given food distribution scenario. The primary focus of this model is the distribution of food, which is a very specific scenario in PKO. EADS[10] Dornier has developed the model to a prototype status, scientifically supported by the University of Passau (Operations Research Department), Germany, and the University of Zurich (Social Psychology Department), Switzerland. A time continuous approach allows the use of differential equations in the PAX model for some of its variables, e.g., fear. Unfortunately, PAX is a proprietary product, and further development will require the assistance of the developers.

Another simulation was developed at the U.S. Naval Postgraduate School (NPS) in April 2002. The "Agent based simulation of German Peacekeeping Operations for Units up to Platoon Level" [Erlenbruch, 2002] is a MAS that follows an event-driven approach and is based on the software package Simkit[11] (also developed at NPS). [Buss, 2001] This model was designed to simulate a generalized scenario in which a unit of peacekeepers has to prevent a mob of demonstrators from reaching a certain area of a historic old town (Prizren, Kosovo). Figure 1 shows the Graphical User Interface (GUI) in which the scenario is displayed. The peacekeepers (leaders and followers) and the

[9] Pax is the Latin word for peace. No other explanation for this name was found in the references. For more information on the use of this model see [Horne, 2003].

[10] European Aeronautic Defence and Space Corporation.

[11] Simkit is a software package for creating Discrete Event Simulation models. The Simkit homepage at http://diana.gl.nps.navy.mil/Simkit/ provides an overview on the package.

demonstrators (followers and leaders; armed and unarmed) are modeled as software agents. Obstacles like rivers, buildings, barbed wire, etc., are software objects, combined to represent the terrain in which the scenario takes place. The user has the opportunity to call for reinforcement (armored personnel carriers or infantry fighting vehicles) on the blue side.

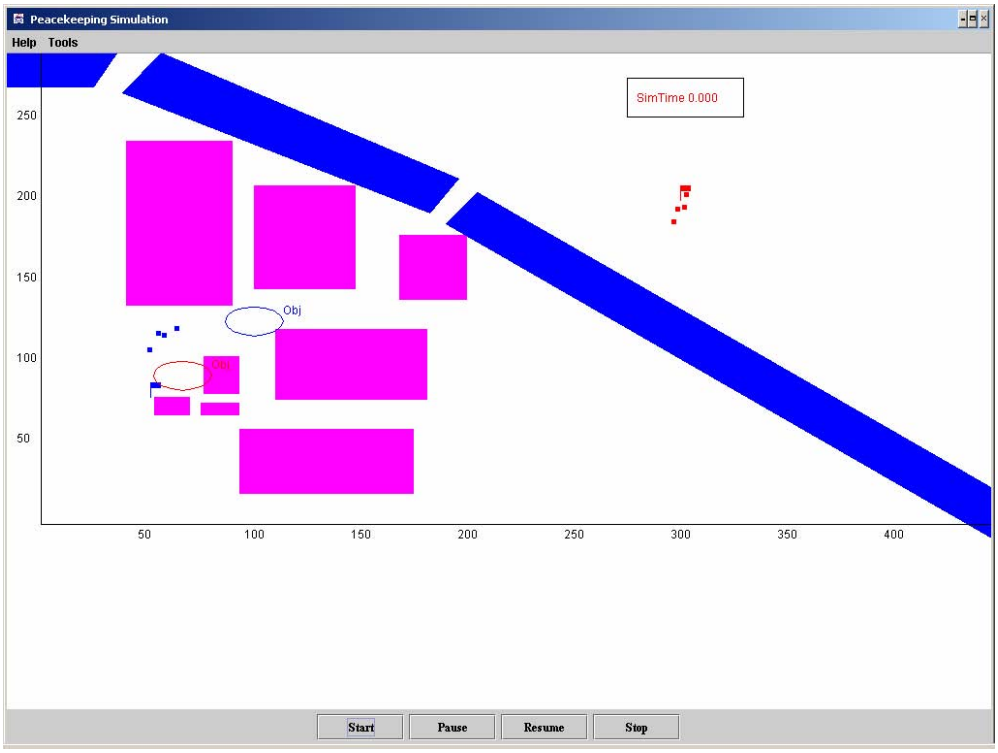


Figure 1. Peacekeeping simulation GUI [From Erlenbruch, 2002]

Figure 1 shows a standard scenario displaying a group of demonstrators, located at (310, 200), who will cross the river on one of the bridges and advance towards their objective at coordinates (70, 90), which is located on a plaza between some buildings. The peacekeepers, located at (60, 120), are advancing to their objective at (100, 130) and will try to keep the demonstrators from reaching their objective. Agents modeling either peacekeepers or demonstrators are characterized by six personality properties (closeness to friendly, obey orders, closeness to leader, affinity to action, risk aversion, and shock influence) and a training level. Each of these properties can take on different integer values and influence the agent's behavior. The current MOE is a weighted sum of the number of protestors who reached their objective, the number of injured persons (blue and red) and the number of people killed (blue and red).

C. RESEARCH OBJECTIVE AND METHODOLOGY

The research objective of this thesis is to obtain a discrete event ABS of PKO in a data-farming environment. This objective builds on previous work, such as Erlenbruch's ABS of German PKO at up to platoon level. Previous models of PKO show that it is possible to model PKO scenarios with ABMs; however, in order to be useful analysis tools, they failed to offer data-farming capabilities. This thesis addresses that gap. Moreover, it proves the model's data-farming capabilities in an exemplary analysis.

The scope of this thesis embraces:

1) The design and programming of relevant software.
2) The development of new MOEs for PKO modeling.
3) The design of an experiment for an exemplary analysis.
4) An exemplary analysis on the agents' personality factors.

The research starts with the design of a discrete event-driven ABS. To realize data-farming capabilities, the software is to follow the model-view-controller paradigm in order to serve as an analysis tool. Erlenbruch's general assumptions can be applied. However, the focus is rather on the model's ability to be data farmed than on user-friendly GUIs. The software includes the design and coding of input and output tools in order to make data farming user-friendly.

One or more new MOEs would be desirable. By their nature, peacekeeping missions are fundamentally different from combat operations; casualties are not always the correct focus. There should be other quantifiable indicators to describe the success or failure of the mission. These MOEs always apply to the specific scenario they are defined for. The question obviously is: How can success or failure of the mission be best quantified? What number—if known—tells us the most about mission success?

As discussed before, it is nearly impossible to simultaneously study the effects of all model factors over their full range (as in a full factorial design). The model presented in this thesis has four different types of agents: leaders and followers, each of which can be red or blue. Each agent has six characteristics (factors) and can be instantiated "one to n" times. Any factor needs to take on at least three levels to identify nonlinear responses.

Eventually, advanced experimental designs may be necessary since it is obvious that full factorial designs cannot be carried out. The exemplary analysis is meant to prove that the model actually is data-farmable and to set a starting point for further research.

Screening techniques going along with advanced statistical techniques of data analysis may generate insight in the nature of PKO. In previous theses, simple descriptive statistical methods, as well as additive regression models, classification trees, neural networks, Bayesian networks, and more proved to be useful techniques to analyze large data sets. The exemplary analysis will use some of these statistical techniques to study the influence of the agents' personality factors on the outcome of simulation runs.

Analysis and critique of the statistical results and a comparison to real-world experience are the final steps taken to—hopefully—add new insight to the nature of peacekeeping missions.

D. ORGANIZATION OF STUDY

The flow of the remainder of this document is as follows. Chapter II explains the design of the simulation model and describes the scenario. Chapter III details the exemplary analysis. It ranges from a description of the experimental design to a portrayal of various statistical techniques. The final chapter, Chapter IV, provides conclusions and recommendations for future work. Appendix A provides a German to English reference. The simulation output is listed in Appendix B and Appendix C provides information on how the reader can obtain the source code of the simulation model.

THIS PAGE INTENTIONALLY LEFT BLANK

II. MODEL DESIGN AND FEATURES

A. OVERVIEW

The bulk of this thesis project presents a simulation that is useful to model peacekeeping scenarios. As in other combat models, a major challenge is to adequately represent the decision-making process of military personnel. However, in MOOTW, the scale of actions and reactions is wider. In addition to military individuals, groups, and units, the model must also be able to capture civilians. Civilians may be armed, but need not be. At times, the civilian side will start out unarmed, but suddenly may use stones or Molotov cocktails.

The range of possible scenarios is too wide to be captured in one simulation model. For this reason, we will design our model in a way that makes further upgrades possible and easy. The first step captured by this work will allow the user to model a scenario where a group of peacekeepers tries to keep a larger group of demonstrators from reaching a certain area that requires the demonstrators to pass them. The model should allow for an escalation of the situation, e.g., peacekeepers firing in the air or shooting demonstrators and demonstrators using their "weapons." A specific description of the scenario that is used in an exemplary analysis is given in Chapter III.

As briefly discussed in Chapter I and previously shown in earlier work, e.g., Erlenbruch's (2002) master's thesis, peacekeeping scenarios can be viewed as complex adaptive systems. Thus, a MAS seems to be a good approach to capture the individual decision-making process and study the outcome of the whole system.

Java is an object oriented programming language and is the computer coding language used in this thesis. The integrated development environment (IDE) used to develop the model, Forte for Java Community Edition, was provided by Sun Microsystems. The simulation model is built using an Application Programming Interface (API) designed by Professor Arnold Buss at the NPS for event-driven simulations called Simkit. For this thesis, Simkit version 1.2.7b (which is a 2003 version) is used. In particular, the simulation presented in this thesis takes advantage of Simkit's mover, sensor, and random number management. For the input

data, i.e., the general setup of a scenario, the eXtensible Markup Language (XML) is used to save load and process the data. JDOM[12] is the API used to operate XML data in Java. During the development phase and, in particular, for debugging and validation purposes, the Actions[13] API provided a set of very useful animation features.

The model is designed to be used by analysts. This chapter provides a description of the model's design and an explanation of the basic ideas that underlie the program code. However, this chapter does not provide a full range of instructions for the use of the program. In order to operate the model, change the basic setup of scenarios, introduce new features, or even upgrade the simulation as a whole, the reader is required to conduct a detailed reading and study of the source code. Because of the upgradeable design, any Java programmer should be able to make further enhancements without changing the basic design of the model. The model is free for all to use. Further information, downloads, and up to date information may be found in Appendix C.

B. GENERAL DESIGN PARADIGMS

This section discusses the model's underlying design paradigms. Alternatives are briefly explained and design choices justified. Since a combination of discrete event simulation (DES) and MAS is still in its early stages, the section starts with a discussion on the pros and cons of DES versus time step simulation (TSS). The next paragraph explains the model-view-controller paradigm (MVCP). A discussion on MOEs is followed by explanations on the upgradeable architecture of the model. An introduction to the management of random numbers when using Simkit concludes this section.

Finally, some hints on notation: Whenever simulation entities are mentioned, e.g., interfaces, classes or objects, they are displayed in *italics*. Classes contain methods. Throughout the text, methods are denoted in Java code notation. For example, there is a method doAssessment(Agent agent). Its name is "doAssessment" and it takes one input parameter of type *Agent*, which has the variable name agent. Variable names are

[12] JDOM is not an acronym. It is, quite simply, a Java representation of an XML document. JDOM provides a way to represent a document for easy and efficient reading, manipulation, and writing.

[13] Animation package developed for Simkit based simulations.

referenced according to their notation in the Java code. For example, sensedEnemies is a variable of type ArrayList in the *BasicPerceprtions* class. Variable names are not displayed in italics.

1. Time Step Simulation versus Discrete Event Simulation (DES)

All simulations have in common that state variables change over the course of a simulation run. For example, a simple multiple server queue model may use Q (the number of customers in a queue) and S (the number of available servers) as state variables. Obviously, both variables change during the course of a simulation run as new customers arrive or customers that received service leave. The question is: "What is the driving force that causes these changes?" There are two philosophies that exist in the simulation world. In TSS, discrete time steps may or may not reveal changes of the state variables. Usually, a time step has a constant value and all time steps are equal. At a time step, the model decides which, if any, state variables change. In a DES, the change of state variables is caused by events.

Regarding the simulation clock, Peter Lorenz from the University of Magdeburg explains the difference between TSS and DES as follows:

> If the simulation clock increases by the same value with every step this is called a time step simulation. It considers equidistant moments and calculates changes of model data only at these moments. If the simulation clock is set forward from event to event we speak of Discrete Event Simulation. An event is the 'sudden' change of the value of at least one model variable. Event and change of states are synonymous in this context. [Peter Lorenz, 2006]

In the area of military applications, TSSs have become widely popular. The underlying philosophy facilitates animation. Military simulations very often model movement processes. For example, a combat scenario can be simulated with a representation of military units moving in a two-dimensional environment. It would be very easy for a computer program to compute the location for any individual moving unit in incremental time steps if their laws of movement are known. Peter Lorenz comments:

> Time step simulation is a widespread construction method for simulation programs. The idea is to divide the simulated time into intervals of the same length and to recalculate all model variables at the end of each of

> these intervals. The time step simulation is a very obvious simulation method. Many 'naïve' programmers, who were not into the matter, have been intuitively using it. An algorithm that checks model variables in intervals for necessary changes is relatively easy to design and implement. [Peter Lorenz, 2006]

ABSs such as ISAAC[14] and MANA,[15] in addition to their time-step approach, take advantage of a checker-board structure for the two-dimensional mover's space. Similar to a pawn in the game of chess, in each time step, one particular mover is allowed to move one square forward, whereas another one could follow the pattern of a knight and move two squares horizontally and one vertically—depending on the mover's speed and destination. Even for a great number of entities, this structure is fortunate for a computer animation. As the simulation clock clicks, the location of any mover is computed and displayed on the screen. Lloyd Brown (2000), using ISAAC for his Master's Thesis on the use of an ABS for human decision-making, explains the model's general design as follows:

> The battlefield in ISAAC is represented on a two-dimensional lattice of discrete sites. Each site of the lattice may be occupied by one of two kinds of agents: red or blue. The initial state consists of either user-specified formations of red and blue agents or a random distribution of red or blue agents. Red and blue flags that represent goals have a user-specified position. A typical goal for both red and blue agents is to successfully reach the flag positioned in the diagonally opposite corner. [Ilachinsky, 1997]

As briefly explained, a time step simulation may have a relatively simple program structure and facilitates animation. As a downside, TSS may be inaccurate for not being capable of representing events that occur within a time interval. This disadvantage can possibly be remedied by minimizing the length of the time steps. Then, however, the simulation may use computing power inefficiently since all state variables are recomputed over and over again even though most will not have changed.

Discrete-event simulation concerns the modeling of a system as it evolves over time by a representation in which the state variables change instantaneously at separate

---

[14] Irreducible Semi-Autonomous Adaptive Combat.

[15] Map Aware Non-uniform Automata.

points in time. These points in time are the ones at which an event occurs. An event is defined as an instantaneous occurrence that may change the state of a system. Although DES could conceptually be done by hand calculations, the amount of data that must be stored and manipulated for most real-world systems dictates that DES be done on a digital computer. [Law and Kelton, 2000]

Again, we listen to Lorenz from the University of Magdeburg as he lectures on this simulation approach:

> Discrete event simulation is a construction method for simulation programs. The simulated time runs through a sequence of moments when discrete events occur. The method is considered to be more efficient, but on the other hand more difficult to handle than time step simulation. It has been given preference in most modern discrete simulation programs. The discrete event simulation divides the process to be emulated into a sequence of events. An event is a change of a process state that is modeled by a sudden change of a model variable. Events are organized in a list that contains the time of each event in ascending order. In this way sub-processes that run parallel are represented with a sequential structure. There maybe any long interval between two events. The selected time unit can be freely chosen and is without influence on the number of calculations.
>
> Difficulties are caused by events that occur at the same time. In large models it is an essential task, to handle events that result from another event correctly. Simulation systems support the user in solving this problem.
>
> Discrete simulation systems today predominantly use the principle of the discrete event simulation. It is more universal than time step simulation. In fact time step simulation can be regarded as a special case of discrete event simulation. [Peter Lorenz, 2006]

Coming back to our multiple server queuing example, the definitions given above should have made clear that, as opposed to only during a time step, an event, say the arrival of a new customer, changes the state variable, in our example Q (the number of customers in a queue), right at the time when the customer arrives. The state variable S (the number of available servers) is also changed in that very moment when a server ends his service to a customer. In case there are further customers waiting in line, the end-service event may cause a start-service event without any delay. As a build-up on

DES, event graphs are a powerful tool to visualize the model. Professor Buss, in his paper "Modeling with Event Graphs," summarizes:

> Event Graphs are a way of graphically representing discrete-event simulation models. Also known as 'Simulation Graphs,' they have a minimalist design, with a single type of node and two types of edges with up to three options. Despite this simplicity, Event Graphs are extremely powerful. The Event Graph is the only graphical paradigm that directly models the event list logic. There are no limitations to the ability of Event Graphs to create a simulation model for any circumstance. Their simplicity, together with their extensibility, makes them an ideal tool for rapid construction and prototyping of simulation models. In this paper we will demonstrate the ability of Event Graphs to leverage simple models into more complex ones with very few additional features. [Buss, 1996]

For a military analyst, it is self-explanatory that a simulation ought to be able to represent an event just in the split-second when it occurs. Furthermore, movers should be able to use any point in a two-dimensional arena. Previous work has proven that it is possible to combine the ideas of ABM with DES. Erlenbruch justifies his choice for an agent-based, discrete event model as follows:

> Discrete event simulation concerns the modeling of a system as it evolves over time by a representation in which the state variables change instantaneously at separate points in time. These points in time are the ones at which an event occurs, where an event is defined as an instantaneous occurrence that may change the state of the system. A time-step simulation, on the other hand, updates all states simultaneously at each processed time step and processes resulting events in a random manner. [Law and Kelton, 2000]

The author chose DES because a pure DES worldview provides more flexibility and modeling power than a pure process-oriented worldview. [Buss, 2001] In an evaluative model, as the one provided in this thesis, it is also important that the state variables change instantaneously, and not at some time step that is chosen for programming purposes. The model must evaluate who acts first and how other entities react to the action. These issues cannot be observed when all state variables are updated simultaneously at a given point in time and the order of action is generated at random.

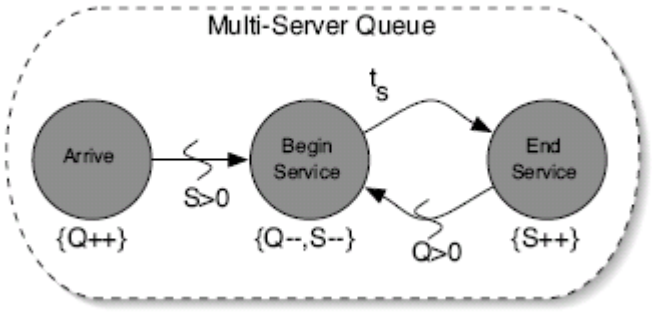


Figure 2. Event graph of a multiple server queue [Buss and Sanchez, 2002]

Figure 2 shows an example of the arrival process represented by the event "arrive." A certain service by one out of a certain number of servers consists of two events: "begin service" and "end service." An arrival event increments the number of customers in the queue, whereas the begin service event decrements it by one. The state variable S (number of available servers) is incremented/decremented by the begin service and the end service events.

Like Erlenbruch beforehand, we will use an ABM in a DES. Peacekeepers and demonstrators will be represented as movers in a continuous two-dimensional (2-D) simulation space. A typical event that occurs is the detection of a demonstrator by a peacekeeper. This event changes one of his state variables: it adds the demonstrator to the list of sensed demonstrators which is a part of the inner environment of the peacekeeper. Based on the situational goal of the peacekeeper and his personality this change in his inner environment may cause action, e.g., firing in the air.

2. Model-View-Controller Paradigm (MVCP)

The MVCP is a software architecture that has a key feature useful for simulation analysts. This architecture allows the user to uncouple the user interface from the model and thus run the model efficiently. A brief and useful definition of this paradigm is given on Apple Computer's Website:

> A common and useful paradigm for object-oriented applications, particularly business applications, is Model-View-Controller (MVC). Derived from Smalltalk-80, MVC proposes three types of objects in an

> application, separated by abstract boundaries and communicating with each other across those boundaries.
>
> Model objects represent special knowledge and expertise, such as a company's data and business logic. Model objects are not directly displayed. They often are reusable, distributed, persistent, and portable to a variety of platforms.
>
> View objects represent things visible in the user interface such as windows, table views, and buttons. A View object is 'ignorant' of the data it displays, as it relies exclusively on the Controller object for data. View objects tend to be very reusable and so provide consistency between applications.
>
> The Controller object acts as a mediator between Model objects and View objects. Usually there is one Controller per application or per window. Controller objects communicate data back and forth between the Model objects and the View objects. A Controller's function is usually very specific to an application, so it is generally not reusable like View and Model objects are.
>
> Because of the Controller's central mediating role, Model objects need not know about the state and events of the user interface, and View objects need not know about the programmatic interfaces of Model objects.
>
> Within the MVC paradigm, enterprise objects are Model objects. By definition, Model objects represent data and business logic. The Enterprise Object technology extends the MVC paradigm so enterprise objects are independent of their persistent storage mechanism. Enterprise objects do not need to know about the database that holds their data, and the database doesn't need to know about the enterprise object formed from its data. [http://developer.apple.com/documentation/webobjects/DesktopApplications/ BasicConcepts/chapter_3_section_12.html, March 2006]

From this definition, there are many advantages of this software architecture. First, when programming such complex software, any programmer will be happy to check the effects of software changes with a GUI. Moreover, a GUI is a desirable debugging tool. The GUI for our model is provided in the Actions API that was developed by Professor Buss, NPS, for Simkit applications. As shown in Figure 13, in Chapter III, agents are displayed as little squares in different colors. Blue agents represent peacekeepers, whereas red agents are demonstrators. The leaders on both sides are indicated by their diamond shape.

The controller is also in the actions API. As shown above, the purpose of the controller is to couple the model with the GUI. In order to get an animation that allows the user to actually see agents move a time-step approach is applied. The controller of the actions API keeps track of all Simkit movers in the model. Therefore, our agents extend Simkit movers. In discrete time-steps, so-called "pings," the controller asks the movers their location, which is an x, y-coordinate in a 2-D simulation space. The location is then given to the "sandbox," which is a representation of the simulation space. On the referring x-, y-coordinate of the sandbox, the referring symbol for the agent is displayed. On any "ping" the location of any agent is updated. By choosing the time-step between two pings small enough, the user will have the impression of continuously moving agents on the GUI.

A model for analysts needs to be data-farmable, meaning it is necessary to do multiple runs with changing parameters. With respect to this requirement, it is necessary for the user to be able to run the model without animation in order to efficiently use computing power. An MVCP architecture allows the user to simply "switch-off" the animation. The model's output, consisting of one or more MOEs, is written to a file as the model iterates one run (or computational experiment) after another.

3. Measures of Effectiveness (MOEs)

A Measure of Effectiveness (MOE) is a quantitative measure that allows insight in the progress and/or success of a process. There are various MOEs in the military world, e.g., force ratio, enemy force attrition, the speed of movements, shot-down ratios, or number of sunk enemy ships.

> During Operation Desert Storm the attrition of SCUDs, tanks and artillery were measured to assess progress in eliminating the enemy's warfighting capabilities. A familiar, and controversial, measure was the body bag count in Vietnam. This experience provides a caution regarding the care that must be taken in developing, choosing, and applying such measures. [John J. Nelson et al., 1996]

Good MOEs are a link between cause (action of own forces) and effect (damage to enemy forces). They tell commanders and decision makers how well they are doing

before the war is over. In this respect, MOEs have some forecasting function, yet they are not always capable of making a reliable contribution to this function.

When it comes to combat modeling, MOEs are used to compare different factor settings. For example, given a certain combat scenario two different tactics are often evaluated by comparing one or more MOEs. For obvious reasons, force attrition used to be a commonly used MOE in combat modeling. In 1914, F.W. Lanchester modeled warfare using a set of differential equations. Their most basic form is based on the assumption that one side's attrition rate is proportional to the other side's number of forces.

> Although the simple Lanchester equations with constant coefficients remain useful for demonstrating some features of combat (e.g., the value of concentrating effort and the associated penalty for breaking up one's forces), especially when it is desirable to do so analytically, they are a poor basis for describing most combat situations. Computer simulations may use Lanchester expressions 'locally' (i.e., for attrition estimates within a given time interval), but the coefficients of those equations change from time step to time step as conditions of terrain, defender preparations, and many other factors change. Good computer simulations recognize that the losing side may choose to break off battle rather than be annihilated.
>
> [http://www.rand.org/pubs/monograph_reports/MR638/app.html, March 2006]

At the dawn of the computer simulation age, criticism of Lanchester equations became louder, although they are still in use for modeling force-on-force scenarios. For their lack of, especially, the "human factor of combat," Dr. Ilachinsky from the Center for Naval Analyses (CNA) claimed that combat scenarios are CAS and thus can be better modeled as ABS.

> From a fundamental standpoint, however, there are many limitations to using LEs to represent modern combat. Two of the biggest limitations are (1) they do not account for any spatial variation of forces (i.e., no link is established, for example, between movement and attrition) and (2) they completely disregard the human factor in combat (i.e., the psychological and/or decision-making capability of the human combatant). Therefore, LE-derived models of land warfare are inadequate for assessing advanced warfighting concepts, such as those being explored by the Marine Corps. In particular, the Lanchesterian view of combat does not adequately

> represent the Marine Corps' vision of combat: small, highly trained, well-armed autonomous teams working in concert, continually adapting to changing conditions and environments. As an alternative, we suggest that recent developments in complex systems theory—particularly the set of multi agent-based simulation tools developed in the artificial life community—provide a new set of tools for addressing land warfare in a fundamentally different way. [Ilachinsky, 1997]

Today, Ilachinsky's ISAAC and other ABSs are widely used by military analysts. ISAAC's visual mode of operation allows an observation of how the battle evolves over time. In addition to this, ISAAC was designed to also be capable of generating quantitative output—i.e., numbers that should tell something about the mission's success.

> To this end, ISAAC provides a capability to (1) generate time series of various changing quantities describing the step-by-step evolution of a battle, and (2) keep track of certain measures of 'how well' mission objectives are met at a battle's conclusion. The former (using built-in statistics measures; see below) yields quantitative snapshots of a battle as it unfolds in time; the latter (using a simple parameter-space mapping technique; see below) yields semi-quantitative measures of 'success' at a mission's end. [Ilachinsky, 1997]

The MOEs Ilachinsky chose fall into seven classes: force sizes; interpoint distance distributions; neighbor-number distributions; enemy-flag interpoint distance distributions; cluster-size distributions; center-of-mass positions; and spatial entropy. Like traditional models ISAAC still uses force ratios, but, in addition to that, this ABS allows insight into many more, particularly location-oriented, MOEs.

Because of the very nature of peacekeeping missions, force ratios, as well as other attrition-based MOEs, are not adequate. As Erlenbruch already pointed out, a killed peacekeeper may demoralize his whole unit and/or demolish the support of their mother country, whereas a dead demonstrator may cause an overall operation failure. Therefore, MOEs for peacekeeping models need to be chosen carefully. In any case, they should be scenario-based. A certain measure that quantifies mission success in one scenario may be completely useless for a different scenario. Erlenbruch used a weighted sum of utility functions. His final MOE was determined by the number of killed and wounded peacekeepers and the number of killed and wounded protestors multiplied by a number that represented how the protestors could advance towards their objective. Like most

combat models, Erlenbruch uses attrition-based measures camouflaged in utility functions in order to generate an overall MOE whose value does not tell the user something intuitively.

The model developed in this thesis is designed to study the influences of human personalities on the arousal of violent behavior in PKOs. Similar to Erlenbruch's work, a group of demonstrators wants to reach a certain location. A smaller group of peacekeepers tries to keep them from getting there by blocking their way. The MOEs should permit an analysis on the question of how different personality factors (explained in detail in Section III.C) have an influence on violent behavior. Therefore, we do not want to conduct a "body bag count." Furthermore, we also don't want the MOE to be determined by the number of demonstrators who pass the peacekeepers. Thus, we will simply measure the time from the first detection of a demonstrator by a peacekeeper, or vice versa, until the first shot is fired. We will also measure the time until the first agent is wounded and the time until the first agent is killed.

Time is a favorable MOE for various reasons. First of all, time can be measured in continuous numbers. This allows for more flexible analysis than discrete numbers do. Time is a common MOE for everyday situations, e.g., sports. Therefore, it is intuitive and easy to explain. Time allows reasonable conclusions for the research questions. If it takes a long time until the first shot the violence potential in the scenario is considered to be low. If the first shot is fired a short time after the first agent was detected, the potential for violence is high.

Using time as our MOE also allows analysis on the question of which one of the agents' personality factors has the greatest effect on a potentially violent escalation of the situation. Simple statistical tests, as well as more sophisticated data analysis models, can be applied.

At this point, the reader may ask: "This is a model for peacekeeping operations. What if there occurs no shot at all? What is the MOE then?" The answer to these questions will be found in Section III.B.

4. Upgradeability

Object oriented programming languages have proven to facilitate programming an ABS. This family of programming languages mirrors complex adaptive systems in the programming world. Building up from primitive type variables, objects may grow up to very complex constructs and end up as software agents. There are three key features that make an object oriented programming language: encapsulation,[16] inheritance,[17] and polymorphism.[18] The first two of these are very important and have to be carefully considered when building an ABS.

Information encapsulation is a standard feature in almost any object oriented program, yet essential for an ABS. Care must be taken: there must be neither any inadvertent information flow nor any outside control of the agents. Software agents are supposed to gain information only through the sensing mechanism or communication. This information is gathered in the class *BasicPerceptions*, which is one major part of their *InnerEnvironment*. The *InnerEnvironment* and the agent's *Goals* determine the agent's actions. Therefore, it would be very unrealistic if an agent would know anything about another agent, except what he sensed or was told.

Inheritance in an object oriented language allows a specific organization of classes. In a tree-like structure, an object can take on any feature of another object and add additional features to it. Therefore, a programmer is able to take advantage of the work others have done before him. In this model, the agent that eventually is instantiated is a *BasicAgent*. A *BasicAgent* extends the *UniformLinearMover* from the Simkit package. This means the *BasicAgent* is a *UniformLinearMover* in the sense that he can do anything that is inscribed in the super class and uses all of its variables. Additionally, the *BasicAgent* has additional variables and features. With the basic idea of inheritance

[16] Encapsulation is the process of hiding all the details of how a piece of software was written and telling only what is necessary to understanding how the software is used. Put another way, encapsulation is the process of describing a class or object by giving only enough information to allow a programmer to use the class or object. [Savitch, 2001]

[17] In a programming language, such as Java, inheritance is a way of organizing classes so that classes with properties in common can be grouped so that their common properties need only be defined once for all the classes. [Savitch, 2001]

[18] In a programming language, such as Java, polymorphism means that one method name, used as an instruction, can cause different actions depending on what kind of object performs the action.

in mind, one can say that any simulation model written in an object oriented language is upgradeable. If one just wants to add capabilities to objects this statement is true. However, one could want to create completely new objects and add them to the framework of the rest of the simulation. Therefore, more sophisticated considerations were made on the software architecture for the model developed in this thesis.

A feature of Java that is even more powerful than inheritance and makes this model upgradeable is the use of interfaces. An interface is a class containing a list of methods that may be implemented by another class. By adding interfaces into a Java program the programmer can build up a framework of "empty classes." Implementations of these empty classes have to contain all the methods listed in the interface and are plugged into the frame the interface created. Additional methods are welcome, but the methods listed in the interface must exist in any implementation of the interface. Now, for a later upgrade it is easily possible to just program another implementation of that interface and use it instead of the first implementation. It will also fit into the plug-in if the class meets the requirement that all methods listed in the interface need to be implemented. For example, in this model there is an interface called *Agent*. It contains a list of getter methods that are useful to be able to access the information that is encapsulated. Furthermore, it has methods like doDetection(), doUndetection(), doWounded(), or doKilled() which are potentially necessary actions in a peacekeeping scenario. Now, the implementation of the *Agent* interface is the *BasicAgent* class. The *BasicAgent* needs to carry implementations of any method listed in the *Agent* interface. If an analyst wants to use a different type of an agent, it would just mean that he would have to program another class, say *EnhancedAgent.*[19] This class must implement the *Agent* interface, e.g., contain any of the interface's methods. Immediately, a scenario with instantiations of the *EnhancedAgent* class may be created without having to change any of the other classes of the simulation. Like *BasicAgent* being an implementation of an interface, any class starting with "Basic," e.g., *BasicGoalManager*, *BasicInnerEnvironment*, etc., are implementations of Java interfaces. Therefore, extensive use of these in this simulation results in a framework that is easily capable of

[19] Potential name for a class representing an upgrade to a *BasicAgent*.

incorporating more sophisticated elements. A detailed description on the simulation program is provided in Sections II.C and II.D.

In many simulation programs, one will find entities that do not originally belong to the simulation model itself. There is no aspect in the real world these entities belong to; they do not represent any scenario aspects. Most of the time, these entities find their way into a model through programming requirements. Sometimes these entities are closely coupled to, if not fully integrated with, the simulation entities. It may be possible that there is an influence through these entities. Therefore, care must be taken that none of these simulation entities are active during simulation runs. One way of taking care of this is the use of tools that help to instantiate classes and then remain passively outside the scenario. These tools are called "factories", "mediators", or "adjudicators."

In our model we use two types of factories: A *WeaponFactory* and an *AgentFactory*. The name Factory is a self-explaining term for what these Java classes are good for: The *WeaponFactory* simply is an outside constructor tool that instantiates the different type of weapons used in the simulation. It returns distinct weapon objects. Of course, *Weapon* is a Java interface whose implementations are different types of weapons. Now, the weapon among other objects is turned over to the *AgentFactory*. In the same way, the *AgentFactory* "puts together" an Agent object and returns it to the scenario. The constructor methods of these Factories are static methods. Therefore, it is not necessary to keep any factory objects in a scenario. However, it is possible at any point in time throughout a simulation run to order a new *Agent* from the *AgentFactory*. That is, a future scenario could include a second group of demonstrators appearing from a different location as well as a peacekeeping leader who would call for reinforcement.

The *AgentCookieCutterMediator* is a class that manages Detection and Undetection events for all agents during a simulation run. Whenever it comes to firings, the *AgentAdjudicator* class is responsible for the evaluation of the effects, i.e., whether agents get wounded or killed. Both classes in their current versions are easily upgraded, extended, or, if necessary, replaced by a more sophisticated mediator.

Another feature that is provided by the Simkit package is the principle of loosely coupled components. This principle takes advantage of so-called "listener patterns" and

makes the entities of a simulation interoperable without explicitly coupling them. Professor Buss explains that in a Simkit model any event is implemented as a user-defined "do" method, e.g., doFiringAt(). A "do" method is simply a method starting with the string "do." Scheduling edges are executed using a method called "waitDelay()." [Buss, 2001] The rest is carried out by the Simkit API using a future event list. Any entity that implements the SimEntity interface can now take advantage of the loosely coupled component principle.

> The mechanism by which two simulation components are linked is the SimEventListener interface, that defines a callback method. An instance of a SimEventListener registers interest in hearing a SimEntity's simulation events with the addSimEnventListener(SimEventListener) method. Whenever a SimEvent occurs for the SimEntity instance, notification is dispatched to all registered SimEventListeners via the callback method processSimEvent(SimEvent).
>
> The behavior of a SimEventListener as implemented in the processSimEvent(SimEvent) method can be completely customized to suit the simulation modeller's needs. Most of the time, the modeller will be content with the default behaviour as implemented in the (abstract) SimEntityBase class. That behviour is that whenever a SimEvent is heard, the object attempts to find a matching 'do' method. If one is found, then it is invoked. If none is found, then nothing happens.
>
> The SimEventListener Pattern is useful in implementing component-based simulation models. [Buss, 2001]

To summarize this aspect of upgradeability, any entity in the simulation is able to interact with other entities without a firm link (reference variable). This principle makes a further upgrade easy to realize. Enhanced entities or those that come in during a simulation run just need to be registered SimEventListener instances. Furthermore, the loosely coupled components principle, as opposed to reference variables, limits agents' access to other agents' variables and methods.

5. Random Number Management

The author of a simulation model that widely uses random numbers for various purposes must ensure that the random numbers are really random. Even if different random numbers from different distributions are required, it is often desirable to pull them all from the same random number stream at first and then transform them into

whatever distribution they belong to. Furthermore, when multiple simulation runs of a specified scenario are carried out it is also highly desirable to use random numbers from a single random number stream for all runs. This means another random number generator must not be instantiated when a new simulation run starts. In order to replicate findings, it must be possible to set a seed. A stored seed ensures the ability to begin a string of numbers from the same starting point in the stream. Again, it is Simkit that provides a sufficient means to establish proper random number management.

> Simkit's design permits much flexibility for generating random variates used in the simulation models. The underlying design goal was to enable the modeller to change any random variate in a model to any desired probability distribution without having to recompile the model. This was to extend to classes generating random variates implemented after the compilation of the original model.
>
> Simkit uses a combination of a RandomVariate interface and an abstract factory that is called to produce instances of the desired implementation using only 'generic' data—that is Strings, Objects, and numbers. [Buss, 2001]

In this simulation various random variates are used. The reaction time of an agent is a uniform variate between 0.1 and 1.0 time units. In certain cases an agent picks a target randomly from a discrete uniform variate. When a scenario is built, agents are distributed randomly around a base (flag). The coordinates are a combination of the base's coordinates plus an error drawn from a normal random variate with a mean of 0.0 and a standard deviation of 20.0. The hit/no-hit and the kill/no-kill decisions are also based on a uniform random variate within the abstract *AgentAdjudicator* class.

The Simkit random package (Simkit version 1.2.7b) offers more random number management options than is necessary to meet the needs of this simulation. In the frame of the RandomNumber interface there are a variety of random number generators that can be instantiated. Changes can even be made on the fly. In order to instantiate them without any undesirable effects on a simulation, the abstract RandomNumberFactory class is used. The default instance that is returned is a Congruential object which is a Linear Congruential Generator that is sufficient for the purposes of this model. Using the getInstance() method the programmer may input a CongruatialSeed[] as a parameter in

order to force the RandomNumber instance to always start at the same spot in the random number stream.

More important than the choice of a distinct generator is the fact that the programmer can use this one and only random number generator as an input parameter to any of the instances of the RandomVariate interface. RandomVariate instances are random number generators for random variates from probability distributions. There are different techniques to cast a U(0, 1) to a random number from a certain probability distribution, say a N(0, 20), such as the inverse transform method, the composition technique, the convolution technique or the acceptance-rejection method. Law and Kelton's book (*Simulation Modeling and Analysis*) provides a good overview on generating random variates. Only one Congruential object exists in the entire simulation model. This one serves as a nucleus for all random variates; thus, the modeler has eliminated the undesirable effects that can occur from the correlation of random numbers.

Furthermore, the Simkit random package is highly flexible and allows further upgrades of the random number organization in many ways. Again, the RandomNumber and the RandomVariate interfaces allow changes even within a simulation run. Furthermore, getter and setter methods[20] make the change of parameters as well as seeds possible at any point in time. With the use of the Simkit random package, there are no limitations for further upgrades concerning the random number management in the model.

C. AGENT DESIGN

This section explains the design of the software agents in detail. Starting from a general overview on the agent's variables and features, we will discuss the main software components that make an agent. The purpose is to explain the functions of the model's main entities rather than discussing the algorithms. A more detailed understanding is

[20] The terms getter and setter methods are often used synonymously for accessor and mutator methods. A public method that reads and returns data from one or more private instance variables is called an accessor method. The names of accessor methods typically begin with "get." A public method that changes the data stored in one or more private instance variables is called a mutator method. The names of mutator methods typically begin with "set."

possible by a look at the source code that is provided in Appendix B. This chapter is meant to explain to the reader what an agent does, how it does it, and why it does it.

1. Agent Variables and Features

As was explained previously, *Agent* is an interface that provides a frame for various implementations. The implementation provided in the model is realized in the *BasicAgent* class.

The *BasicAgent* is inherited from the Simkit *UniformLinearMover* class. It is designed in a building block construct. It consists of an *InnerEnvironment*, which is also designed as an interface. The implementation of the *InnerEnvironment* in this model is the *BasicInnerEnvironment*. The building blocks that make the *BasicInnerEnvironment* are three interfaces: *Properties*, *Perceptions* and *History*. *Properties* and *Perceptions* are implemented in *BasicProperties* and *BasicPerceptions*, whereas an implementation of *History* is left for further upgrades.

Implementations of the *Agent* interface also consist of a goal apparatus. The management of the goals of a *BasicAgent* object is done by the *GoalManager*, which is a Java interface implementing a *BasicGoalManager*. The *BasicGoalManager* holds and manages the five goals a *BasicAgent* possesses. All of these goals are implementations of the *Goal* interface.

Furthermore, *BasicAgent* has a *Weapon* variable. *Weapon* also is a java interface. There are seven different implementations of the *Weapon* interfaces starting from *WeaponMG3* (representing a 7.62 machine gun) to *WeaponStone* (representing a stone, which can be used as a weapon by some demonstrators).

Given this building block construct, it is possible for a programmer to easily enhance the *BasicAgent* without making a lot of changes to the *BasicAgent* class. As an example, one could simply add one or more goals. Of course, the programmer would have to write the classes for the additional goals. Then, he could write an *EnhancedGoalManager*[21] (which could be an inheritance from the *BasicGoalManager*). Unfortunately, the GoalManger is instantiated in the *BasicAgent*'s constructor. So, the

[21] Possible name for a GoalManager in an upgraded model.

last change one has to make would be to instantiate the *EnhancedGoalManger* instead of the *BasicGoalManger*.

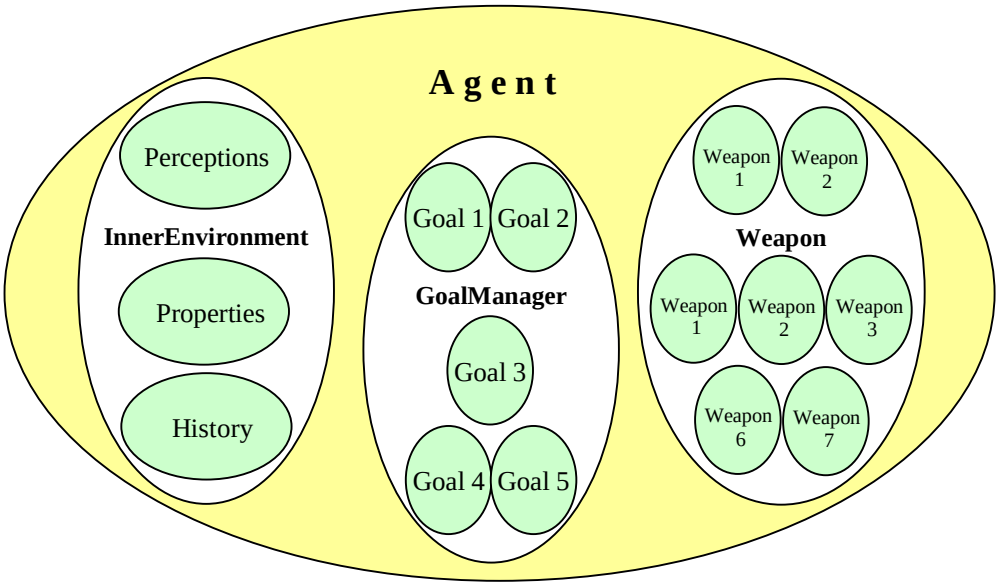


Figure 3. Building block structure of the *Agent* interface's implementations

Figure 3 depicts the building block structure of the possible implementations of the *Agent* interface. Any of the ovals could be any implementation of an interface. Thus, any of them may be taken out of the structure and be replaced by a different implementation without changing the structure. Also, there is space enough to add additional implementations.

A *BasicAgent* is characterized by its side (red or blue), its rank (leader or follower) and its health (healthy, wounded or killed). These three variables, however, are kept by the class *BasicProperties*, as are the personality factors. The *BasicAgent* uses getter methods to access these variables.

The *BasicAgent*'s variables that are related to the moving capabilities, like location and maxSpeed, are kept in the super class which is the Simkit *UniformLinearMover*. The *BasicAgent* does not yet have a PathMoverManager. For a scenario as simple as ours it is not necessary to use a PathMoverManager. However, if

the model should be upgraded with infrastructure (buildings, streets, rivers, bridges, etc.) it may be necessary to have a PathMoverManager that is able to navigate the Agent around the obstacles.

A *BasicAgent* has a single sensor. However, the variable mySensor is of type Sensor. Sensor again is an interface out of Simkit. Simkit also provides a set of Sensor implementations. Out of these we picked the CookieCutterSensor and extended it. Therefore, the *BasicAgent* uses the *AgentSensor* as an implementation of the Sensor interface. Since the Detection and Undetection events cause action on the agent level, the sensor also has to be kept on the agent level.

The *BasicAgent* class has a variety of getter and setter methods. With respect to encapsulation of information, the use of such methods is a highly sensitive matter. Care must be taken that other agents will never change nor have access to the variables which are not meant for them to know or change. Since it was not possible to renounce them completely, these methods had to be reduced to a minimum. In particular, the use of setter methods is very sensitive. Setter methods allow changing encapsulated information. However, since some of the information is kept not on the *Agent* level, but on the *Properties* level, and is also subject to change, a small number of setter methods was inevitable.

The most interesting methods, those that represent the decision-making process as well as the actions of a peacekeeping soldier (or a demonstrator), are the “do” methods. As was mentioned before, the “do” methods take advantage of the SimEvent Listener pattern in order to manage the events. In this manner loosely coupled components can interact with each other. The rest of this chapter provides a brief explanation on the “do” methods of the *BasicAgent*.

In the real world, there are times when persons change their mind and try to achieve something different than what they wanted to before. Usually, this happens after an assessment of their situation. In a peacekeeping scenario, a demonstrator may try to reach a certain area. All of a sudden, he hears a rifle being fired somewhere. He would assess his situation and may change his mind. He may stop, turn around, and flee. The

model initiates all of these change of goals with the doAssessment(Agent agent)[22] method. Whenever an agent is about to accomplish one of his goals or is subject to a situational change the doAssessment(Agent agent) method is invoked. This may happen when the agent detects a crowd of demonstrators, when he loses eye-contact with his leader, when a rifle is fired, when an agent gets wounded or killed, etc. The doAssessment(Agent agent) method, with the help of the nextAction() method, causes the agent to stop. Then, all scheduled events of this agent are deleted from the event list. Then, an algorithm determines his next action based on his situational awareness, his properties (personality factors), and his currently most important goal. The next event for this agent is then scheduled on the event list.

The method doDetection(AgentContact contact) is designed to represent a situational awareness change. For example, all of a sudden, a peacekeeper may see one or more demonstrators approaching him, i.e., at a certain distance they enter his detection range. The cookie-cutter principle that is underlying the detection process in this model is a reasonable approximation of reality. This principle is based on the assumption that within a certain range anyone is recognized. Whenever another agent is within the sensor's range the *BasicAgent* knows it is there. There is no way of approaching an agent without being detected as soon as entering its sensor range.

In a further upgrade the effect of different sensing mechanisms could be tested. For example, one could implement a probabilistic sensing mechanism. The closer two agents get to each other the more likely it is that they detect one another. In a peacekeeping scenario, the detection of someone may cause an action. Say, all of a sudden, a young soldier in his first peacekeeping mission finds himself surrounded by a crowd of yelling demonstrators. That may cause him to try to get back to his unit or to use his weapon and fire in the air, based on his panic level. Therefore, the doDetection(AgentContact contact) method invokes an assessment process which is carried out in the doAssessment(Agent agent) method. The parameter in the doDetection(AgentContact contact) is an *AgentContact* object. This class is designed to hide information. A *BasicAgent* in his *Perception* object does not maintain a list of all

[22] Java notation for a method: methodName(parameterType parameterName).

the agents that were detected, but of the *AgentContacts*. The *AgentContacts* listed in the sensedEnemies or the sensedOwn ArrayLists serve as a filter and provide only that potion of information that the *BasicAgent* is supposed to know, e.g., its location, whether it belongs to the *BasicAgent*'s side, whether it has a weapon, and whether it is firing. A more detailed view of the management of sensed agents is given in Section II.C.4.

In the real world, it may be that the distance between a peacekeeper and a demonstrator gets too far for them to maintain their awareness of each other. Again, the cookie cutter principle models this with sufficient accuracy in our scenario. If an agent gets out of the sensor range of another agent they lose awareness of each other. The doUndetection(AgentContact contact) method carries out this event. Sensed agents are taken from the sensedEnemies or the sensedOwn ArrayLists in the *BasicPerceptions* class. There is but a little memory function. If the broken contact is the leader of his own forces, this agent will remember the exact location where this contact breaks.

In an escalating real-world peacekeeping scenario, peacekeepers may use their weapon. For example, they may fire in the air or at demonstrators in self-defense. The methods doFiringAir() and doFiringAt() cover these cases. This method decreases the amount of rounds in the weapon. It is very obvious that in a real-world scenario the noise of a shot will change the situation. Therefore, other agents must be able to hear the one that fires in the air. This mechanism is carried out by two methods; both of them are invoked via the SimEvent listener pattern. The doFiringAir() causes a "bang" event. The doBang() method in the *AgentAdjudicator* class then is followed by a "hear shooting" event for other agents that are located within a specified range. The respective doHearShooting(Agent agent) method belongs to the *BasicAgent* class. Furthermore, a shot in the air in the real world will cause a peacekeeper to assess the situation. Did he reach his goal? Does the crowd draw back from him? Is he in control again? Consequently, the doFiringAir() causes the doAssessment(Agent agent) method to be invoked.

The doFiringAt() method covers all the effects of the doFiringAir() method, however, there is more to it. It represents aimed fire at a person, which results in either a miss or a hit. If it is a hit, it may result in a kill or a wounded. Before this, a target needs

to be picked. It is obvious that aimed fire in a self-defense situation is fired on someone who is likely to be using a weapon himself, whereas in a panic situation the target may be picked at random. At first, the method counts the number of sensed enemies; for example, the length of the sensedEnemies ArrayList in the *Perceptions* object. It then checks for "enemies" who are firing and enemies who have a weapon. Then, a target is picked according to the following priorities: enemies that are firing, enemies that have weapons, randomly. The method then invokes the doAdjudicate(AgentDuel duel) method in the *AgentAdjudicator* class. This method is the kernel of the abstract *AgentAdjudicator* class. It has an algorithm that determines whether it was a miss or a hit and whether the hit causes a kill or a wound. The algorithm takes into consideration the training level of the agent, the distance, and the type of weapon. Like the doFiringAir() method, doFiringAt() will result in a doAssessment() in the end.

It is the *AgentAdjudicator* class that cares for the statistical data. In any simulation run, the simulation time of the first shot, the first agent wounded, and/or the first agent killed is measured and written in an output file.

A *BasicAgent* may get wounded or killed by the aimed fire of other agents. The injury of an agent is carried out by the doWounded(Agent agent) method. This method simply changes the state variable health, which is managed by the *BasicProperties* class. Moreover, it changes the properties (personality factors) of the agent. For example, a soldier who gets wounded may be under the influence of shock, he may try to keep closer to his own group, he may try to avoid further risks, and he probably will not have the same affinity to action anymore. Such property changes are carried out by the doWounded(Agent agent) method. Furthermore, the model assumes that a person who gets wounded for the second time is killed. For an analysis of the durations until the first shot, the first agent wounded, and the first agent killed this assumption seems reasonable. However, if the model is used for different purposes this assumption may need to be reconsidered.

Even in MOOTW, peacekeepers as well as civilians may get killed. The representation of a kill is carried out by the doKilled(Agent agent) method. This method deletes all future events of this agent from the event list. It changes the state variable

health. Furthermore, a killed agent is taken from the ArrayLists of the sensed agents in order to make sure that a killed agent is not considered to be a threat anymore. As previously explained, the statistical data collection is done by the *AgentAdjudicator* class.

The *BasicAgent* class also has some methods that are not captured by the SimEvent listener pattern. Strictly speaking, these methods contain code that is carried out by the "do" methods. In order to keep the "do" methods small and to make this code available for various methods and to keep the program in good order it was packed into independent methods.

The following UML diagram (Figure 4) provides an overview on the primary variables and methods of the *BasicAgent* class.

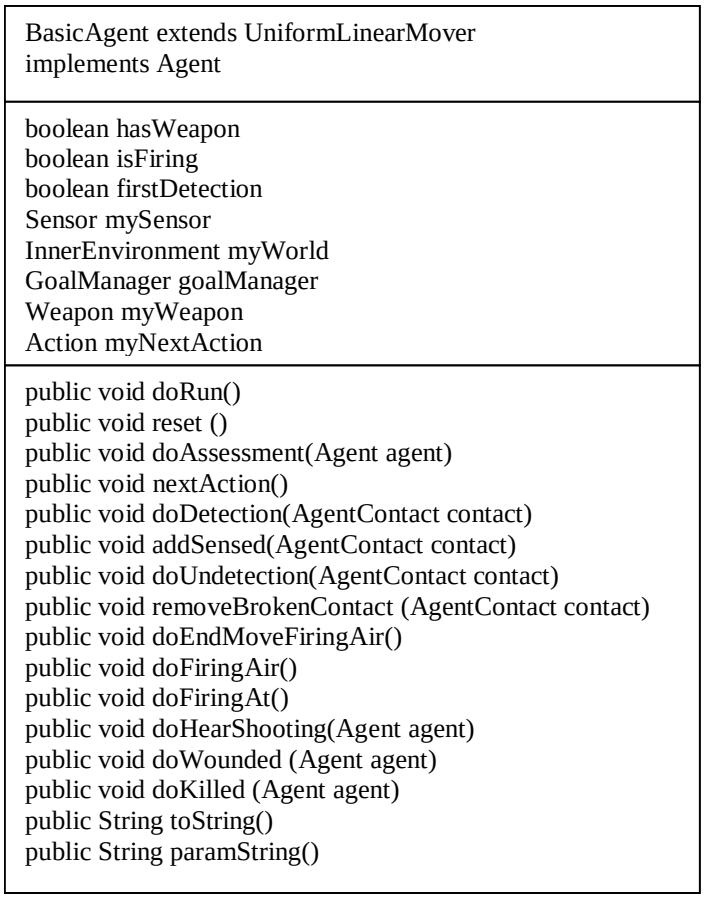


Figure 4. UML diagram of the *BasicAgent's* variables and methods

2. The Inner Environment

Previous work has proven a distinction between an outer and an inner environment to be a useful principle. [Erlenbruch, 2002] In this context, the outer environment would be the representation of a real world peacekeeping scenario. The outer environment includes anything that exists. It is an objective glimpse of the whole scenario. On the other hand, the inner environment only exists inside an agent. It is the representation of the agent's perceptions of the outer environment. For example, when peacekeepers and demonstrators intermingle it could be the case that a peacekeeping soldier loses eye-contact with his leader. The leader is still there, but the soldier does not see him anymore. In the model space this would mean that the leader still is a part of the outer environment. However, the same leader would no longer be a part of that agent's inner environment. Erlenbruch explains the relationship of the outer and inner environment in his model as follows:

> The author uses the idea of detectors, effectors, and the performance system as a filter from Holland's agent model. The agent model used in this thesis, the PKO-Agent, has a detector to perceive the outer environment and to build a picture of the perceived outer environment to build an inner environment. This inner environment is the knowledge of the outer environment the agent has. All agent decisions are based on the inner environment. The PKO-Agent also has effectors, to react to the outer environment. It furthermore has a performance system that filters the 'fittest' reaction out of a set of possible reactions, depending on the inner environment. [Erlenbruch, 2002]

In general, the outer environment is perceived through one or more detectors. Inside the agent, the changes of the inner environment may cause actions of the agent. The links from the agent to the outer environment (of which it is a part of) are the agent's detectors and effectors. Of course, an agent would not act based only on his inner environment. Between the inner environment and the agent's effectors there is a more or less sophisticated decision-making apparatus. For the moment, we call it the performance system. Its functions will be explained later in this chapter. It is important to keep in mind that the information within the inner environment causes the performance system to make a decision on if and how to react. An agent that takes on any action is a change in the outer environment. This change of the outer environment (which is always

the same for all agents) may be sensed by some other agents, thus causing a change of their inner environment. This may cause actions to be taken by these agents. This, in turn, may cause changes in the outer environment. Referring to Holland's agent model, Erlenbruch depicts this loop in Figure 5.

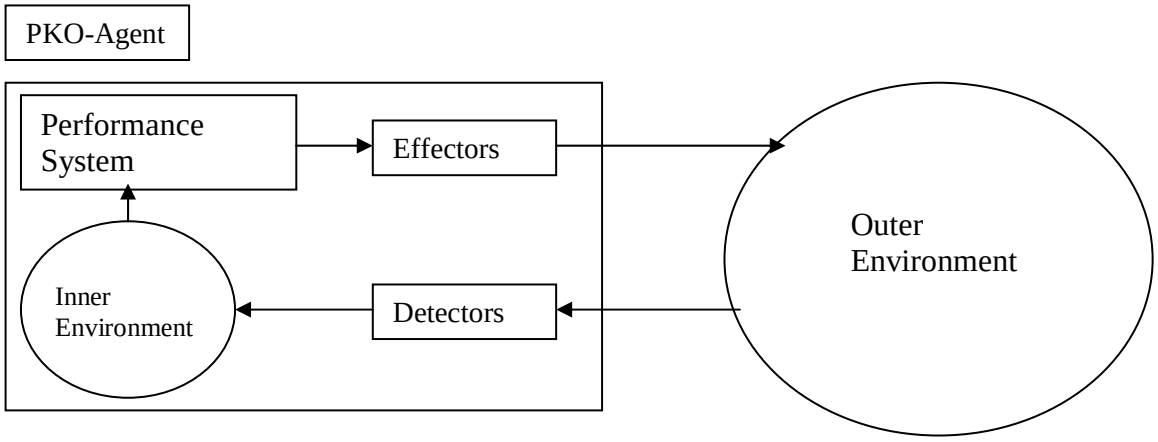


Figure 5. PKO-Agent model depending on Holland's reactive agent [Erlenbruch, 2002]

In our model, the inner environment is more than just a subset of the outer environment. As shown in Figure 3, *InnerEnvironment* is a Java interface. The implementation of this interface is the class *BasicInnerEnvironment*. Its three main variables are of type *Perceptions*, *Properties*, and *History*. In addition to these, *BasicInnerEnvironment* also holds two variables of type Point2D, the objective and the base. In order to keep the architecture of the model in good order this class serves as a layer between the agent level and the subordinate classes.

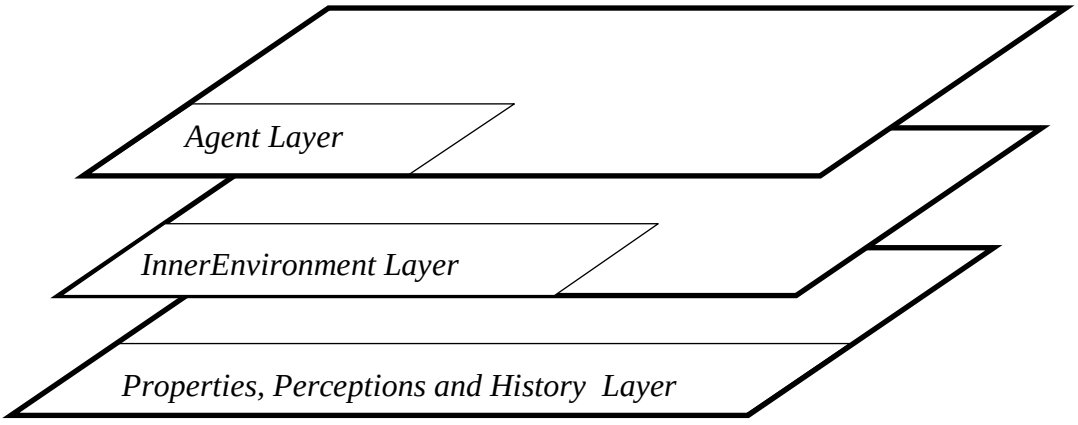


Figure 6. Layer model of the *BasicAgent*

Figure 6 is a depiction of the different layers of information in a *BasicAgent*. On each layer information is kept and processed. The links to the outer environment, detectors and effectors, are located on the agent layer. So are the various components of the performance system. Information of sensed agents, however, is located on the perceptions layer. Obviously, the inner environment builds an intermediate layer. Any information that is kept on the perceptions layer and is required on the agent layer must be passed through the inner environment layer. In this model, there is no filter function implemented. Information will always be passed through in an unaltered manner.

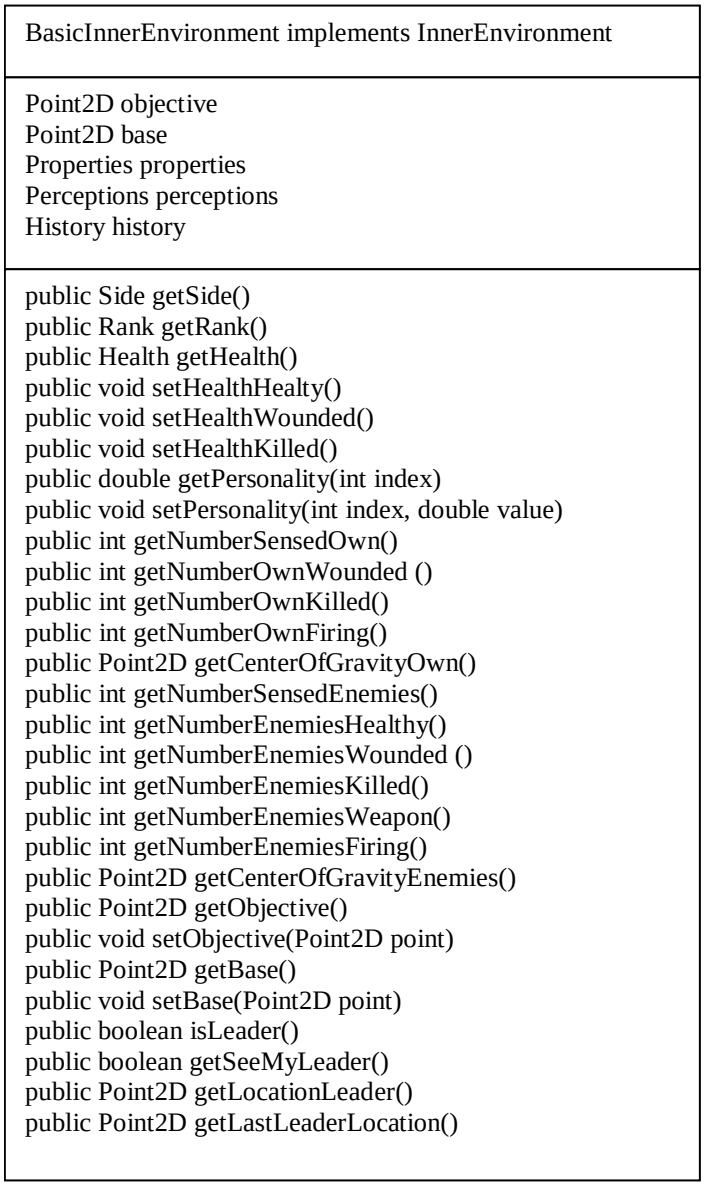


Figure 7. UML diagram of the class *BasicInnerEnvironment*

Figure 7 provides an overview on the variables and methods of the *BasicInnerEnvironment* class. As previously described, "getter" and "setter" methods pass accurate information. It is left for future work to add dizziness to the methods. For example, if a peacekeeper has sensed a group of demonstrators they may be 10, 11, or 12 persons. Adding a tolerance from a random variate would be a simple add-on to the existing model.

3. Agent Properties and Personality

In the real world, various players within a peacekeeping scenario are different. There may be military personnel as well as civilians. Some may possess the role of a leader, others followers. As the scenario develops, originally healthy people may get wounded or killed. People may be cautious, or behave like daredevils. Some young soldiers may try to stay close to their leader, others may be relatively independent or even adventuresome; some may act strictly according to given orders, while others remain more flexible.

All the individual properties of a *BasicAgent* are managed by the class *BasicProperties*. The variables mySide, myRank and myHealth pertain to each *BasicAgent* in the peacekeeping scenario. The variable mySide is allowed to take the value "blue" or "red." Following the standard military notation, blue agents represent peacekeepers and red agents represent demonstrators. The variable myRank is allowed to take the value "leader" or "follower." There is not a big difference between the capabilities of a "leader" and a "follower." Leaders in the scenario are the focal points for their groups. Followers want to stay in touch with their leaders. The model does not include giving and receiving orders at runtime, as this is not necessary for the simple scenario that is being modeled. Such an orders mechanism using, e.g., doGiveOrder(Order) and doReceiveOrder(Order) methods, is left for further upgrades to the model. Both mySide and myRank will keep their values throughout a simulation run. The variable myHealth, however, may change from the state "healthy" to "wounded" or "killed" over the course of a simulation run. The use of the classes *Side*, *Rank* and *Health* to manage the values which are static and final—meaning they are

constants that do not require objects—has proven to be more stable than variables of type String.

The variable myPersonality is a double array of length 7 that holds the so-called personality factors. Following Erlenbruch's approach, personality factors are defined as:

1) Closeness to friendly: it defines how important it is for the agent to stay close to friendly agents.
2) Obey orders: it classifies how carefully the agent follows the given order to advance the objective.
3) Closeness to leader: it characterizes how important it is for the agent to stay close to its leader.
4) Affinity to action: which defines the agent's affinity towards action.
5) Risk aversion: which describes how hard the agent tries to avoid risks.
6) Shock influence: which characterizes how high the influence of an observed casualty is for the agent. [Erlenbruch, 2002]

The seventh value in the array is reserved for the agent's training level. The training level influences the agent's reaction time and its probability of being hit. Personality factors can take on values of type double between 0.0 and 10.0. All personality factors are subject to change during a simulation run. Therefore, personality factors do not represent a person's character, but rather a person's attitude in a certain situation. Change of personality factors is carried out in the agent layer. For example, when the method doHearShooting(Agent agent) is invoked, the factor "obey orders" is decreased, whereas the factors "risk aversion" and "shock influence" are increased. Also, when the method doWounded(Agent agent) is invoked, the factor "affinity to action" is decreased while the factors "closeness to friendly," "risk aversion," and "shock influence" are increased. These factors determine the agent's course of action. This mechanism will be explained in Section II.C.5. For the moment, it is sufficient to understand that an agent possesses a set of goals. In any situation, one of these goals is the most important. The most important goal determines the *BasicAgent*'s next action. Personality factors have an immediate and direct effect on the selection of the most important goal.

For review, the object *BasicInnerEnvironment* contains two major components: *BasicProperties* and *BasicProperties*. With the above explanation in mind, it is fair to

say that the object *BasicProperties*, as part of the *BasicInnerEnvironment*, is designed to manage all the factors organically belonging to the *BasicAgent*. The reader may recall that in this model, agents represent humans in a peacekeeping scenario. Therefore, it is fair to think of the variables and methods of the class *BasicProperties* as human factors in the simulation. The other major part of a *BasicInnerEnvironment* is the object *BasicPerceptions*. Figure 8 provides an overview on the variables and methods of the class *BasicProperties*.

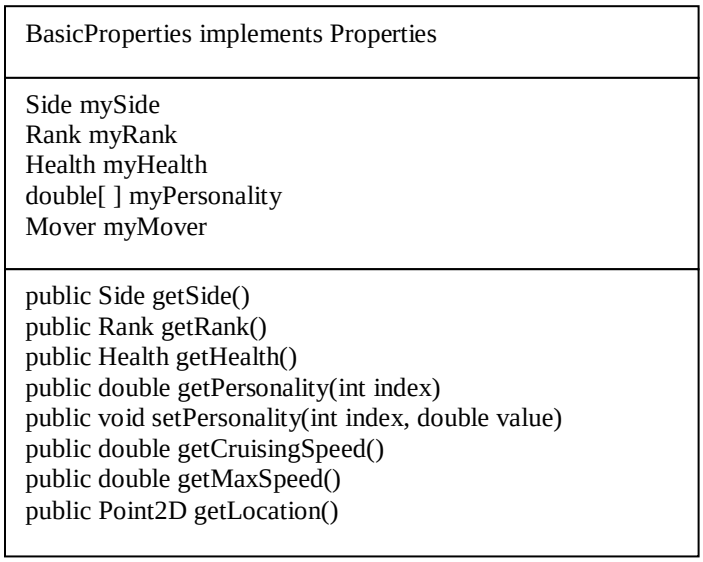


Figure 8. UML diagram of the class *BasicProperties*

4. Agent Perceptions

The class *BasicPerceptions* is designed to manage and operate data that can be classified as an agent's perceptions. It is the representation of a peacekeeper's or a demonstrator's awareness of his environment. The heart of the class *BasicPerceptions* consists of a set of two Java ArrayLists—"sensedOwn" and "sensedEnemies"—that count detected agents. For encapsulation purposes, *BasicAgents* are not stored in ArrayLists, but are transformed into *AgentContact* objects. The class *AgentContact* only allows a certain amount of data to be accessed via its getter methods. As was mentioned before, information is not modified as it is saved or returned. For example, as the method getNumberOfSensedEnemies() is invoked, it will return the true number of AgentContacts that are stored in the sensedEnemies ArrayList. There is no fuzziness in the model at this point. It is left for further work to add fuzziness to the information

management process. Instead of processing the modifications on the perceptions layer, information could be altered as it is passed through the innerEnvironment layer when the information is stored, when the information is returned, or both ways. Since the model currently does not include obstacles such as buildings, the third ArrayList, named sensedObstacles, remains an empty variable.

The following brief summary shall guide the reader through the model's assumptions. At first, the class *BasicPerceptions* provides methods to add and to remove contacts. Whenever a different agent is picked up by the *AgentSensor*, an *AgentContact* object is put into the respective ArrayList. For any decision-making process, there are methods that return the number of sensed agents from both sides: the number of those that have weapons, the number of agents that are firing, and the number of agents that are wounded or killed. Furthermore, a peacekeeper may possess the desire to get physically closer to his fellow peacekeepers. As military personnel open fire, demonstrators may want to run away from the danger zone. In both cases, the location to reach is determined by an agent's perception of "where the majority is"—either his own or the enemy's. In order to model these intentions, there are methods that return the center of gravity of all sensed contacts in either list. The possibility exists to determine a new waypoint for the mover and thus invoke the method doStartMove(). Furthermore, it may be important to a young soldier to stay close to his leader. In case eye contact is lost, human intuition seeks to look first at the very location where the leader was last seen. The method getLocationLeader() will return the location of the leader as long as he is in the agent's sensor range. In case the leader was removed from the sensedOwn ArrayList, the method setLastLeaderLocation(Point2D location) helps to memorize the location where the agent spotted his leader for the last time. The method getLastLeaderLocation() returns this location. Figure 9 provides an overview of class *BasicPerceptions*' variables and methods.

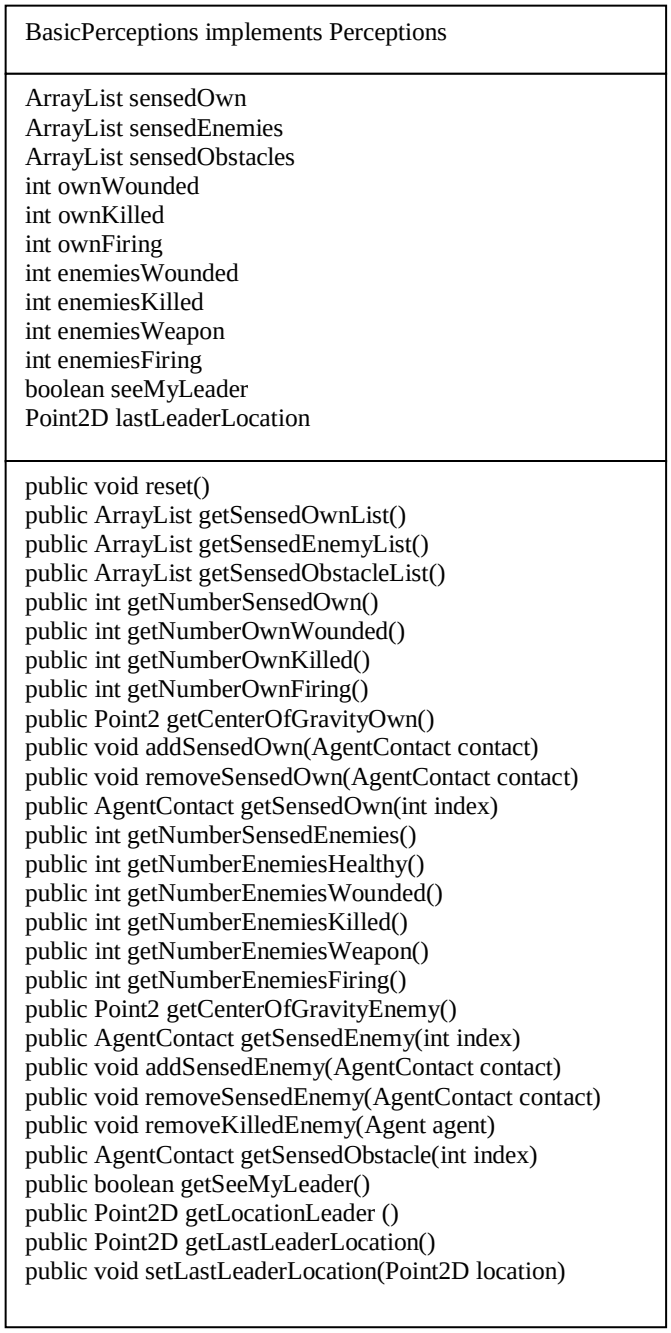


Figure 9. UML diagram of the class *BasicPerceptions*

5. Agent Goals, Tickets, and Management

For review, Figure 5 depicts the exchange of information between a *BasicAgent*'s *BasicInnerEnvironment* (with its two main elements: objects, *BasicProperties*, and *BasicPerceptions*) and the outer environment. Besides detectors and effectors, this loop also includes an apparatus called the performance system. The purpose of this chapter is

to explain the agent's performance system. The heart of this performance system is the goal apparatus. Erlenbruch has shown that it is useful for PKO agents to have a set of goals and a mechanism to organize and manage:

> The goal structure defines the agent's desires. The desires depend on the agent's personality and the environment, as the agent perceives it. Out of a given number of goals the agent always tries to achieve its most important goal. This is one of the agent's ways to adapt to the perceived environment. [Erlenbruch, 2002]

Following Erlenbruch's idea, it appears appropriate for this simulation to operate using five goals. Each goal is organized in its own Java class. All of these classes are implementations of the *Goal* interface and are extended from a *BasicGoal*. The *BasicGoal* class consists of all the variables and methods that are used by all goals. Specifically, the *BasicGoal* has one variable of type *TicketManager*, which is an interface.

Figure 10 provides a breakup of the agents' performance system into goals and tickets. Based on Professor Hiles' general design of agents, tickets are representations of possible actions:

> Each goal has a certain number of tickets, which generate the action the agent will take to respond towards the outer environment. […] As the goals, these tickets also have associated weights and a measurement function to evaluate the weight value. [Erlenbruch, 2002]

Goals are representations of a person's desires and tickets are actions associated with a goal. To realize a person's desires, there may be various actions that are promising. Any goal requires a ticket manager apparatus, which in turn makes the decision of what action to carry out.

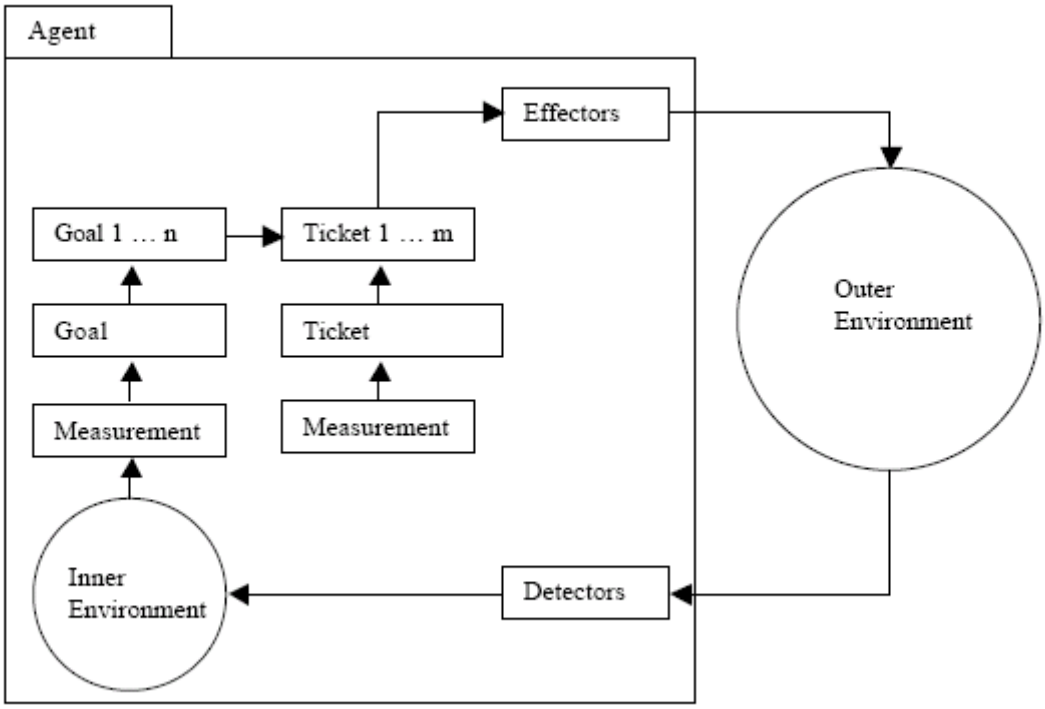


Figure 10. Hiles' agent model [Erlenbruch, 2002]

The heart of the class *BasicGoal* is the method updateSituation(). This method represents an evaluation carried out by a peacekeeper or a demonstrator in a certain situation and computes ratios such as:

1) ownRatio: The number of sensed own agents relative to the total number of sensed agents.
2) enemyRatio: The number of sensed enemy agents relative to the total number of sensed agents.
3) distanceToObjectiveRatio: The distance to the objective relative to the initial distance to the objective.
4) distanceToLeaderRatio: The distance to the own leader relative to the sensor range.
5) distanceToFormationRatio: The distance to the center point of the sensed own agents relative to the sensor range.
6) distanceToEnemyRatio: The distance to the center point of the sensed enemy agents relative to the sensor range.
7) shock: A weighted sum of wounded and killed agents on both sides relative to the total number of sensed agents

All of the factors listed above are stored in a vector called mySituationalState. This data structure serves as a numeric expression for an agent's situational evaluation. Together with the personality factors, mySituationalState is then used to compute individual goals.

Each individual goal is represented by a subclass of *BasicGoal*, thus adding its own variables and methods to the superclass.[23] For the simplified scenario, where a group of peacekeepers are encountered by a larger group of demonstrators, one single ticket associated to each goal is sufficient. Therefore, in this special case the goal and the associated ticket are closely coupled. Any of the tickets is a class that implements the *TicketManager* interface. This way, the model is flexible for future upgrades. Instead of a single ticket, a complete ticket manager apparatus (similar to the goal manager apparatus) may be implemented.

Having explained the idea of goals and tickets in general, the five goals of the *BasicAgent* and their associated tickets are:

1) *GoalAdvanceObj* and *TMAdvanceObj* represent an agent's desire and action to move toward its objective. In this simulation the objective of an agent is a location. If this goal is active, it represents the agent's primary desire to move to this location. Like any other goal it has an algorithm to compute the goal's score in order to have the *BasicGoalManager* choose the primary goal of the agent. *GoalAdvanceObj* also has a method that computes the next waypoint for the agent. With the design choice to abandon a path mover manager, the algorithm that decides on the next waypoint is best implemented in the goal. This goal, like the other goals, has a method to compute the speed at which the agent then moves to the next waypoint.

2) *GoalBackToFlag* and *TMBackToFlag* represent a person's desire and action to move towards his base. All agents receive a second location called the "base" from the input data set. The base is the agent's starting point and at the same time represents a secure area to withdraw to. In the event this goal has the highest score, it will compute the next waypoint for the agent so that it will move back to its base. Also, the goal has a method to compute speed of movement.

3) *GoalBackToFormation* and *TMBackToFormation* represent an agent's desire to move closer to its own group. In order to compute the next waypoint the class takes advantage of the method

[23] When the principle of Inheritance is applied in object oriented programming, classes extend their superclasses. For example, *BasicAgent* extends its superclass *UniformLinearMover*.

getCenterOfGravityOwn() in the *BasicPerceptions* class. Instead of randomly selecting an agent and moving towards his location, the next waypoint will be at a location that is closer to the majority of the sensed agents. Similar to other goals, the method getTicketSpeed() computes a moving speed that applies to the agent's situation and personality.

4) *GoalBackToLeader* and *TMBackToLeader* represent a person's desire to move closer to his or her leader. This goal regularly gets a high score whenever the leader of one side leaves the sensor range of an agent. Based on the personality factors, it may not get the highest score, but if it does, the goal becomes the active one. The method getNextWayPoint() then computes a waypoint in the direction where the agent last sensed the leader. The moving speed is then computed according to the other goals.

5) *GoalEngageEnemy* and *TMEngageEnemy* represent an agent's desire to act forcefully or even violently towards other agents. Based on the situation, this goal and its associated ticket make the decision whether to fire in the air or to fire on another agent. There are only these two choices at this point. Therefore, it could be handled by only one ticket. In case an upgrade should implement other actions under this goal, e.g., "push the enemy aside" or "keep hold of an enemy," the implementation of a ticket manager and multiple tickets is recommended.

The class *BasicGoalManager*, which is an implementation of the *GoalManager* interface, is designed to manage these individual goals. The variable goalKeeper of type *Goal* holds all goals. The method determineHighestGoal() calls all goals to return their current score and returns the index of the goal with the highest score. Finally, the method getNextAction() calls for the active goal to return the *Action a*ccording to its ticket. *Action* objects are storage devices for the agents' actions, designed in order to increase the stability of the simulation. An *Action* object is an information triplet that consists of an action name, an action destination, and an action speed.

Figure 11 depicts the complete structure of the *BasicGoalManager*. It shows that the *BasicGoalManager* class instantiates all of the goals. They are kept in the variable goalkeeper, which is of type *Goal* [ ].[24] Therefore, the *BasicGoalManager* is flexible enough to incorporate more goals as well as others. Then, any of the goals instantiates its individual ticket (instead of a ticket manager that would be capable of managing multiple tickets for a single goal).

[24] The notation *Goal* [ ] means an array of objects of type *Goal*. Recall that *Goal* is a Java interface.

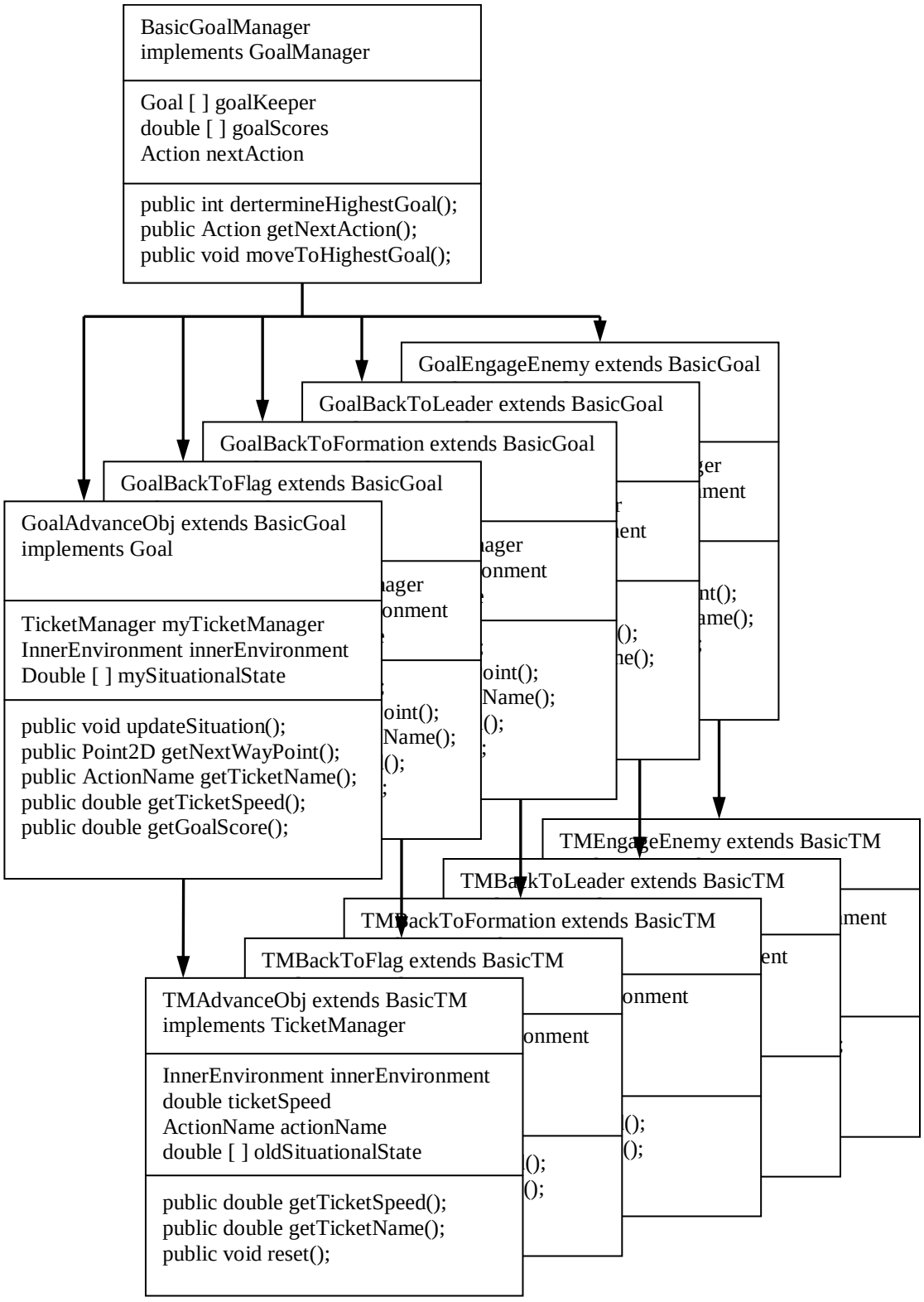


Figure 11. UML diagram of the BasicGoalManager, Goals, and Tickets

D. SCENARIO DESIGN

This chapter explains the scenario which is the basis for the analysis. Generally speaking, the scenario should be as slim as possible and as rich as necessary in order to answer the research questions. Since the model was not developed from scratch, but built up on previous work, it made sense not to start with a completely new scenario. This chapter provides a summary of previous work and explains the two tools used to store and to display a scenario—XML files and the 2-D animation.

1. Previous Work

The scenario in this analysis builds upon work conducted by Erlenbruch. He generalizes an excerpt of a map exercise used in the German Army for leadership training in the context of peacekeeping missions:

> The map exercise takes place in PRIZREN, KOSOVO. German peacekeeping forces have been deployed to PRIZREN as part of the UN KFOR forces. The following extract of an order and three situation developments describe the current situation and the task for one company, 3./ Einsatzbataillon 1 of the TASK FORCE PRIZREN, which has been deployed to PRIZREN. The KPC and PBP, which are mentioned in the order, are groups that fought in the war and that are now illegal terror organizations. [Erlenbruch, 2002]

For his simulation, Erlenbruch extracts one scenario, generalizes it, and programs a complete animation from scratch. He transfers the urban terrain of the town of Prizren, Kosovo into the2-D representation shown in Figure 12.

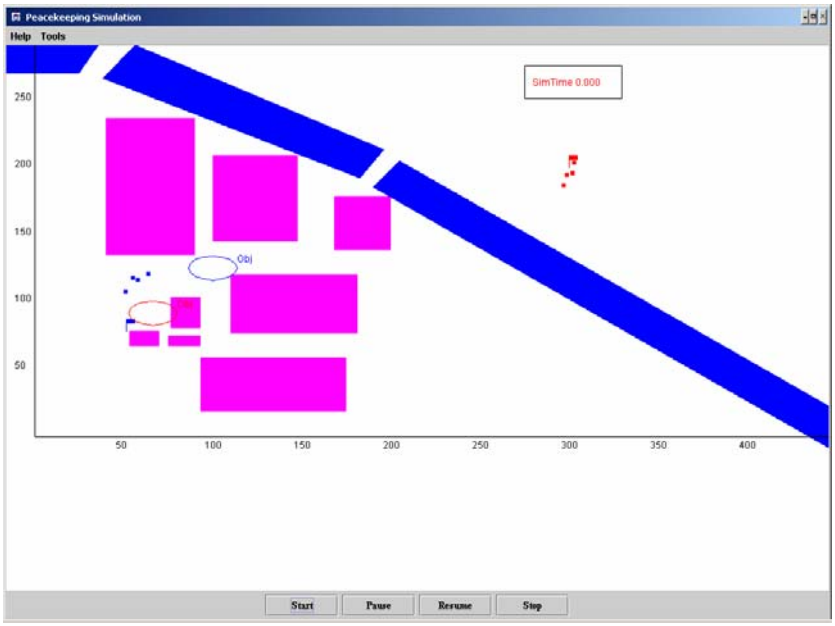


Figure 12. Generalized PRIZREN environment [Erlenbruch, 2002]

For more information on the above scenario, the reader may reference Chapter III of Erlenbruch's thesis.

2. The Outer Environment in its 2-D Animation

The focus of this thesis is analysis of encounters between peacekeepers and demonstrators. Urban terrain representations (buildings, roads, rivers, etc.) usually require a computationally intensive line-of-sight algorithm. The simulation presented in this thesis assumes encounters takes place on a spacious plaza. Interactions of influence between personality factors and arousal of violence are explored. Simulating urban terrain consumes a tremendous amount of computing power without adding additional knowledge to the research questions explored in this thesis.

Our MOEs focus on the time differences between the following events. As explained in Section II.B.3, three different MOEs can be measured during the simulation runs:

1) timeToFirstShot – timeToFirstDetection

2) timeToFirstWounded – timeToFirstDetection

3) timeToFirstKilled – timeToFirstDetection

Implicitly, there is one more MOE that may result from the other three: it may happen that no kill occurs over the course of a simulation run (sometimes not even a shot). In these cases, there is no data output, as will be shown in Chapter III.

Figure 13 (best viewed in color) shows the animation of the outer environment. For Simkit based simulations, particularly simulations that use *Movers*, the actions package provides a set of classes that build an animation. This animation was very useful during the development, debugging, and verification of the model. The programmer could actually observe the movements of the *BasicAgents*.

At this point, the animation does not display actions other than movements, e.g., agents firing, getting wounded, getting killed. Incorporating symbologies for these actions is left for future work. In case the model is used for analysis techniques that exceed the statistical methods applied in this thesis, e.g., course of action development, it may be highly desirable to have such a sophisticated animation.

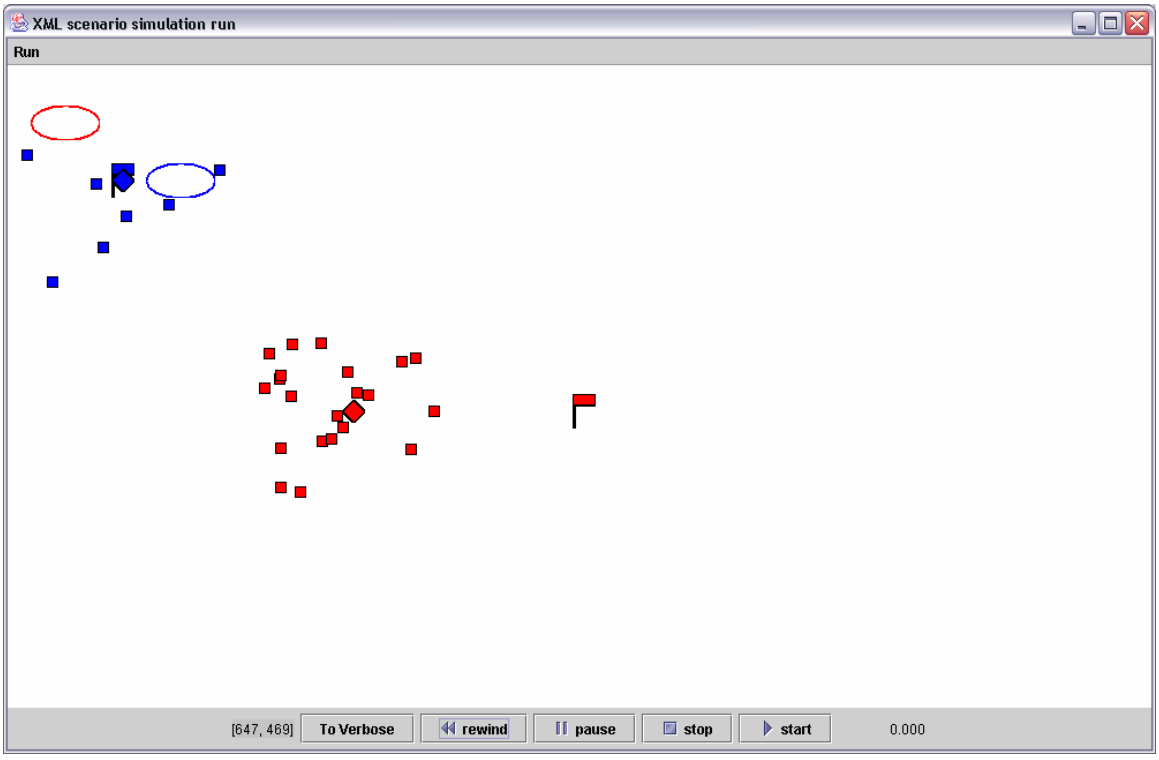


Figure 13. Scenario for simulation in a 2-D animation

In Figure 13, the small squares represent humans—the blue ones are peacekeepers, the red ones are demonstrators. Diamonds represent leaders. These

symbols are fixed to the underlying movers and visually move when the simulation begins. Flags represent the base of each side and the ellipses are the objectives. As the reader can observe, the group of peacekeepers are at their objective, whereas the demonstrators need to pass the peacekeepers on their way towards the red objective. As soon as the peacekeepers sense demonstrators (and vice versa), the events begin to escalate—to include possible firings in the air and firings at person entities.

3. The Use of XML Files

Among other applications, XML technology is used to model data for computer processing. For a brief background explanation on this technology the following summary is taken from Magrolis' (2003) thesis, which also includes XML files as data input tools for a simulation model on aircraft operational availability.

XML is platform and language independent, open source, license free, and has international standards. XML technology addresses how to represent data and surrounding information to describe its content and form, thereby enhancing the data's meaning. For example, sections in a newspaper are differentiated by their spacing and position on the page and the use of different fonts for titles and headings. XML works much the same way, but uses symbols instead of spaces and fonts [Ray, 2001].

If an input file has no boundaries or labels, then a program cannot possibly know how to treat a piece of text and distinguish it from any other piece. A newspaper without spaces and only one font style is a large and uninteresting block of text. A computer program would not be able to distinguish where a particular article began or ended. XML solves this problem [Ray, 2001].

XML is not a computer language, but a standard for creating markup languages (i.e., tags of element names) that meet XML criteria. In other words, XML describes a syntax that can be used for individuals to create their own unique markup language. Individuals are able to create their own set of "tags" in which to describe their data when using XML [Hunter, 2001].

Another value to using data stored in an XML document is that it can easily be checked for errors prior to being used in a computer program. To do this, a schema was

developed to validate the data prior to its use. A schema is a “template” that sets the data requirements and provides a way to define the XML document. As an XML-based language is used for structuring XML documents, the schema describes constraints that govern the order and sequence of data and specifies permissible value spaces for all data used inside the document [Kim, 2003].

If a required data element is missing, the schema will not validate the XML document. Something as simple as an extra “space” character in the input file can ruin a simulation run. Not only could an unwanted space character ruin the simulation, for large files it could take hours, if not days, to locate the error. When validating an XML document with a schema, the schema will find the errors for the user, thus saving hours (possibly days) of needless troubleshooting.

Figure 14 displays a portion of one of the XML documents created for this thesis. Note that there are special symbols called “markup” or “tags.” The tag <BlueLeader> is called a start tag, and the tag </BlueLeader> is called an end tag and they define the beginning and end of a collection of text. That is, they act as bookends marking the beginning and end of data. Each set of “tags” and the data in between them are collectively called an “element” [Ray, 2001].

```
<Scenario>
        <Name>FirstXMLScenario</Name>
        <BlueForces>
                <BlueLeader>
                        <Side>Blue</Side>
                        <Rank>Leader</Rank>
                        <Personality>
                                <CloseToFriendly>1.0</CloseToFriendly>
                                <ObeyOrders>1.0</ObeyOrders>
                                <CloseToLeader>1.0</CloseToLeader>
                                <AffinityToAction>10.0</AffinityToAction>
                                <RiskAversion>1.0</RiskAversion>
                                <ShockInfluence>1.0</ShockInfluence>
                                <Training>1.0</Training>
                        </Personality>
                        <InitialLocation>
                                <XValue>100.0</XValue>
                                <YValue>-100.0</YValue>
                        </InitialLocation>
                        <Base>
                                <XValue>100.0</XValue>
                                <YValue>-100.0</YValue>
                        </Base>
                        <Objective>
                                <XValue>150.0</XValue>
                                <YValue>-100.0</YValue>
                        </Objective>
                        <Sensor>
                                <Range>105.0</Range>
                                <Type>AgentSensor</Type>
                        </Sensor>
                        <Seeds>
                                <ReactionSeed>123456</ReactionSeed>
                                <TargetSeed>654321</TargetSeed>
                        </Seeds>
                        <Weapon>
                                <Name>P8</Name>
                                <Capacity>8</Capacity>
                                <MaxRange>50.0</MaxRange>
                                <Deviation>0.08</Deviation>
                                <RoundsPerMinute>7.0</RoundsPerMinute>
                        </Weapon>
                </BlueLeader>
```

Figure 14. Sample Portion of XML document

Figure 14 shows how XML data is combined into a hierarchical structure. The items in < > relate to each other in parent/child relationships. For example, the elements <CloseToFriendly>, <ObeyOrders>, <CloseToLeader>, <AffinityToAction>, <RiskAversion>, <ShockInfluence>, and <Training> are all children of the <Personality> element, which is a child of the <BlueLeader> element. Data in Figure 14 are referenced

by their element name, thus making the querying of data less prone to errors. That is, information formatted according to XML standards is "self-describing" [Hunter, 2001]. Looking at the data, the reader can easily locate the information pertaining to, e.g., "Weapon." The Element <Weapon> consists of five child elements listed between <Weapon> and </Weapon>. When a scenario is created and the <Name> of the <Weapon> is required, in this case "P8" is returned.

While Figure 14 shows a sample data structure in an editor window, Figure 15 depicts the "grid view" of the same file in an XML development environment, such as Altova XML Spy. Doing a copy-paste, it is very easy to change an existing scenario, e.g., putting additional Red Followers in it. To change the data is even easier—the actual values are just keyed in to their cells.

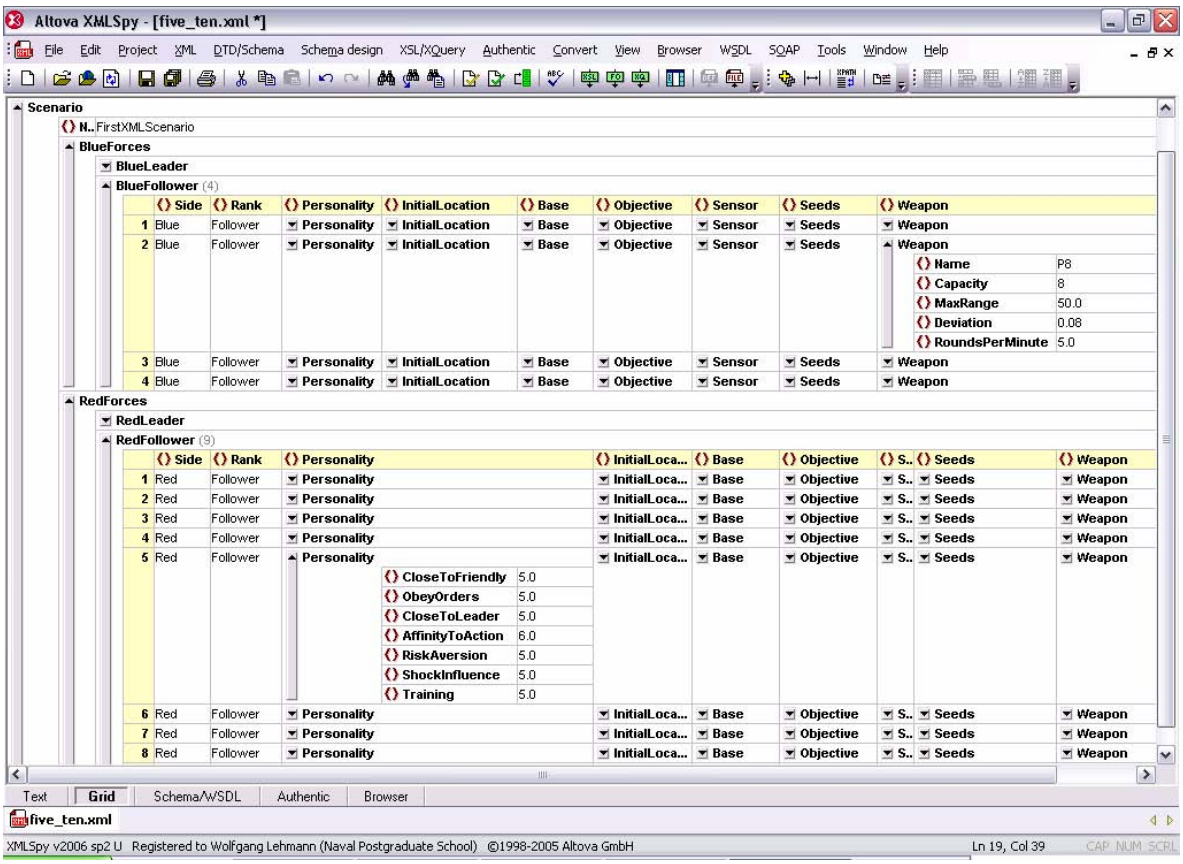


Figure 15. Scenario for simulation in an XML IDE (Altova XMLSpy)

THIS PAGE INTENTIONALLY LEFT BLANK

III. EXPERIMENTS, RESULTS, AND ANALYSIS

One of the main reasons why simulation is such a successful technique in operations research is that it allows analyses of systems and processes. Using simulation one may study the effects on a variation of treats without the necessity to carry out live experiments, which may be time-consuming, expensive, dangerous or even impossible. Figure 16 shows a depiction of a generalized input-output relationship of a simulation model. In our case, the simulation model represents a real-world peacekeeping scenario. X is multidimensional and symbolizes the space of possible input factors, e.g., the number of peacekeepers, the number of demonstrators, etc. Y is the range of possible outcomes. The figure suggests there is a relationship between X and Y. An analyst is interested in finding this relationship in order to draw conclusions for the real-world system the simulation represents. Specifically, if the simulation model is an ABS, it is the surprise, the unexpected, the analyst is searching. Any surprise will result in further explorations. The discussion of the phenomenon with experts of the system may then lead to conclusions, no one would have thought of.

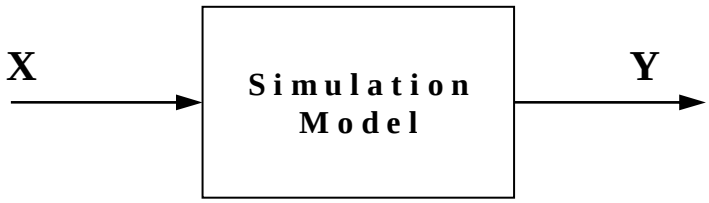


Figure 16. Input-output relationship of simulation models

However, care must be taken to make sure that valid conclusions are drawn from the results of a simulation model. Many people make a huge effort on designing and developing a simulation model. Analysts must pay close attention in inputs and outputs if they wish to draw valid conclusions from a simulation model. Law and Kelton criricize treating a simulation study as only an exercise in computer programming.

> In many simulation studies a great deal of time and money is spent on model development and programming, but little effort is made to analyze the simulation output data appropriately. As a matter of fact, a very common mode of operation is to make a single simulation run of somewhat arbitrary length and then to treat the resulting simulation estimates as the 'true' model characteristics. Since random samples from

> probability distributions are typically used to drive a simulation model through time, these estimates are just particular realizations of random variables that may have large variances. As a result, these estimates could, in a particular simulation run, differ greatly from the corresponding true characteristics for the model. The net effect is, of course, that there could be a significant probability of making erroneous inferences about the system under study. [Law and Kelton, 2000]

The simulation model itself was explained in detail in Chapter II. This chapter deals with the input and output of the simulation. On the input side, a design of the experiments that were carried out is described. Then, the second part of this chapter deals with a quantitative analysis of the output.

A. DESIGN OF EXPERIMENT (DOE)

The space of possible scenarios of the model is extremely wide. For example, the model allows for a huge variation in numbers of agents. There may be one or more groups of agents for each side, i.e., "red" or "blue." Each group of agents may have one or more "leaders." Each group of agents may start from different locations in the 2-D animation space, may have a different base, a different objective, etc. Of course, the total number of agents is limited by the available computing power. As a rule of thumb, the model in its current design allows for a total of up to 100 agents. Because of the exponentially increasing number of detections, undetections, and decision-making processes, a single run without animation on a modern PC (Pentium 4, 2.66 GHz, 512 MB) may last up to three hours, depending on the number of agents, their starting conditions, and the simulation stopping criteria.

The agents themselves may have different personality factors, different training levels, different weapons, and different sensors. They may start at different locations and advance towards different objectives. These parameters may be all homogeneous, groupwise homogeneous, or completely heterogeneous.

So far, it should have become obvious that it is impossible to explore all input parameters. Even the exploration of the six personality factors plus the training level in a two-level full factorial design would lead to a total number of $2^7 = 128$ experiments.

Each experiment requires its own XML input file. The output of each experiment will also be stored in an extra text file.

While Figures 14 and 15 showed portions of the input in XML notation, Table 1 depicts the complete information of one exemplary XML file. As explained earlier, even the number of groups of agents or the number of leaders or followers in each group may be variables, and thus be subject to some sort of "tactical analysis." In this table, we have a group of eight peacekeepers (one of them is a leader) and three times as many demonstrators. The personality factors are groupwise homogeneous. Again, each nongrey cell of this table can be considered as an input variable.

expample scenario				Personality Factors							Initial Location		Base		Objective		Sensor		Weapon				
		Side	Rank	CloseToFriendly	ObeyOrders	CloseToLeader	AffinityToAction	RiskAversion	ShockInfluence	Training	xCoord	yCoord	xCoord	yCoord	xCoord	yCoord	Range	Type	Name	Capacity	MaxRange	Deviation	RoundsPerMinute
Agent Number	1	Blue	Leader	5.0	5.0	5.0	5.0	5.0	5.0	5.0	100.0	-100.0	100.0	-100.0	150.0	-100.0	100.0	AgentSensor	P 8	8.0	50.0	0.1	3.0
	2	Blue	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	100.0	-100.0	100.0	-100.0	150.0	-100.0	100.0	AgentSensor	G 36	30.0	500.0	0.1	6.0
	3	Blue	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	100.0	-100.0	100.0	-100.0	150.0	-100.0	100.0	AgentSensor	G 36	30.0	500.0	0.1	6.0
	4	Blue	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	100.0	-100.0	100.0	-100.0	150.0	-100.0	100.0	AgentSensor	G 36	30.0	500.0	0.1	6.0
	5	Blue	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	100.0	-100.0	100.0	-100.0	150.0	-100.0	100.0	AgentSensor	G 36	30.0	500.0	0.1	6.0
	6	Blue	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	100.0	-100.0	100.0	-100.0	150.0	-100.0	100.0	AgentSensor	G 36	30.0	500.0	0.1	6.0
	7	Blue	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	100.0	-100.0	100.0	-100.0	150.0	-100.0	100.0	AgentSensor	G 36	30.0	500.0	0.1	6.0
	8	Blue	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	100.0	-100.0	100.0	-100.0	150.0	-100.0	100.0	AgentSensor	G 36	30.0	500.0	0.1	6.0
	9	Red	Leader	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	P 8	8.0	50.0	0.1	3.0
	10	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	P 8	8.0	50.0	0.1	3.0
	11	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	P 8	8.0	50.0	0.1	3.0
	12	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	P 8	8.0	50.0	0.1	3.0
	13	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	P 8	8.0	50.0	0.1	3.0
	14	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	P 8	8.0	50.0	0.1	3.0
	15	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	Stone	3.0	10.0	1.0	1.0
	16	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	Stone	3.0	10.0	1.0	1.0
	17	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	Stone	3.0	10.0	1.0	1.0
	18	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	Stone	3.0	10.0	1.0	1.0
	19	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	Stone	3.0	10.0	1.0	1.0
	20	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	Stone	3.0	10.0	1.0	1.0
	21	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	Stone	3.0	10.0	1.0	1.0
	22	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	Stone	3.0	10.0	1.0	1.0
	23	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	Stone	3.0	10.0	1.0	1.0
	24	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	Stone	3.0	10.0	1.0	1.0
	25	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	Stone	3.0	10.0	1.0	1.0
	26	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	Stone	3.0	10.0	1.0	1.0
	27	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	Stone	3.0	10.0	1.0	1.0
	28	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	Stone	3.0	10.0	1.0	1.0
	29	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	Stone	3.0	10.0	1.0	1.0
	30	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	Stone	3.0	10.0	1.0	1.0
	31	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	Stone	3.0	10.0	1.0	1.0
	32	Red	Follower	5.0	5.0	5.0	5.0	5.0	5.0	5.0	300.0	-300.0	300.0	-300.0	50.0	-50.0	90.0	AgentSensor	Stone	3.0	10.0	1.0	1.0

Table 1. Complete information space of simulation input files

For the scope of this thesis, we show that the model is data-farmable with an exemplary analysis. Out of the complete input space, we pick three personality factors and carry out a two-level full factorial design. Therefore, $2^3 = 8$ experiments are carried

out. The research question for this exemplary analysis is: Which of these three input factors has the most significant influence on the result of the simulation runs?

In order to focus on the three chosen variables, it is indispensable to keep the other factors constant. In other words, the experiments take place in a scenario consisting of a constant and a variable portion.

The constant portion of the scenario is described as follows:

1) There are 8 blue and 24 red agents.
2) Each group has one leader; the rest are followers.
3) For the blue group, the initial location is at their base (100.0, –100.0);[25] their objective is slightly shifted to the right (150.0, –100.0). The leader starts exactly at the initial location. The followers are scattered randomly around the leader.
4) Likewise, for the red group, initial location and base are at (300.0, –300.0) and their objective is at (50.0, –50.0). This way, the red group, advancing towards their objective, needs to pass the blue group.
5) The sensor type is *AgentSensor*. For the blue agents, the range is 100.0, whereas the red agents have a range of 90.0.
6) The blue leader is armed with a pistol of type P8. Blue followers have the German standard infantry rifle G36.
7) In the red group, only the leader and five other agents are armed with the Pistol P8. When the scenario escalates during a run, the rest of the group may use stones as weapons.
8) All agents (red and blue) have the same personality factors. Therefore, we have homogeneous groups, both are equal.

The variable portion of the scenario focuses on the personality factors. "CloseToFriendly," "AffinityToAction," and "RiskAversion" shall be the variable factors, whereas "ObeyOrders," "CloseToLeader," "ShockInfluence," and "Training" are kept constant. To give a short review of Section II.C.3, "Training," together with the other personality factors, is stored in the array "myPersonalities" of the *BasicProperties* class. The values of these personality factors range from 0.0 to 10.0. Therefore, we define the constant level to be 5.0, whereas the low level is 1.0 and the high level is 10.0.

[25] Coordinate notation in parentheses, meaning (x-coordinate, y-coordinate). The y-coordinate is negative because the origin of the coordinate system is in the top left corner of the animation window.

Table 2 shows the two-level full factorial design as an extract from the simulation's information input space. The letters "C," "L," and "H" symbolize the different settings: constant (5.0), low (1.0), and high (9.0).

		Personality Factors						
		CloseToFriendly	ObeyOrders	CloseToLeader	AffinityToAction	RiskAversion	ShockInfluence	Training
Scenario	S1	L	C	C	L	L	C	C
	S2	L	C	C	L	H	C	C
	S3	L	C	C	H	L	C	C
	S4	L	C	C	H	H	C	C
	S5	H	C	C	L	L	C	C
	S6	H	C	C	L	H	C	C
	S7	H	C	C	H	L	C	C
	S8	H	C	C	H	H	C	C

Table 2. Experiment design for three input factors

Each of these scenarios requires its own XML input file. Likewise, the output of each scenario will be written to its own text file. The simulation program is capable of reading in a list of output files successively without having to restart the simulation again and again. In order to gather a sufficient number of data points, each scenario is run 100 times. Therefore, the total number of data points for the analysis is:

8 scenarios * 100 runs = 800 data points

To give a short review of Section II.B.3, the analysis focuses on if there is a kill and, given a kill, the time until the first kill. In order to eliminate possible effects of the personality factors on the moving speed, especially of the red group, the time until the first detection is subtracted. Therefore, the analysis focuses on the escalation of the situation; an imaginary stopwatch is started as soon as the first agent detects an agent from the other side and is stopped when the first agent is killed.

B. OUTPUT ANALYSIS

Thorough output analysis is the reason why we do simulation in operations research. As explained before, the design of a model, as well as the design of the experiments, has a primary purpose: To apply statistical methods to the output data so that we can glean insights into the real-world system represented by the simulation. At the end of the day, if one has not gained some new knowledge or insight from his simulation, what was the whole effort good for?

This chapter provides an exemplary output analysis. The purpose is to show how the model can be utilized in order to draw conclusions and gain a deeper insight into the type of peacekeeping scenarios that are simulated.

At first, a very general overview of the output data is provided. As it turns out, there is a remarkable quantity of simulation runs in which not even a single kill occurs. In these runs, the MOE (timeToFirstKill – timeToFirstDetection) can not be calculated. Therefore, we will apply a statistical method to the complete output data that allows for a binary classification. The question here is: how strong is the influence of the three variable input factors on the occurrence of kills? The statistical approach that is applied is the technique of classification trees. Finally, the data is filtered and further analysis is carried out under the premise "given a kill." In this case, a linear regression model on the three main effects and their interaction terms is built.

Further analysis is left for future work. As an example, it would be interesting to explore all seven personality factors. We recommend defining more than two levels for each. Consequently, a three-level full factorial design would mean $3^7 = 2{,}187$ scenarios. For each scenario, the model must run a sufficient number of times. As explained in the previous chapter, each scenario requires its own XML input file. Each scenario will also generate its own output file. Therefore, it is highly recommended to apply more efficient experimental designs, such as Latin Hypercubes, to extended analyses.

The exemplary analysis in this chapter provides some data on the computing time required. Specifically, 100 runs of a single scenario (8 blue agents, 24 red agents) on an average PC (Pentium 4, 2.66 GHz, 512 MB) lasts about 30 minutes. Of course, the

duration of a run also depends on the stopping criteria of the simulation. For this exemplary analysis, the simulation is stopped after two agents are killed, five agents are wounded, or after 200 time units.

For future analysis, it is highly recommended to do test runs, taking advantage of the animation, at first. With the experience of these, the stopping criteria of the simulation can be adapted.

1. General Overview on Output Data

Appendix B, at the back of this thesis, provides a table of the output data. Originally, the simulation program saved the output of each scenario separately in a text file. The table in Appendix B still has a block structure so that one can easily determine the scenario and the run number for each data point. The first column (fD) lists the timeToFirstDetection, the second (fS) shows the timeToFirstShot, and the third (fK) contains the timeToFirstkill. Finally, the fourth column contains a "finished"-statement for each run, including the run number. The reader may count 100 runs for each block, e.g., scenario. This sums up to a total of 800 data points.

The first observation to make is that during some runs we don't obtain a kill; sometimes not even a shot. Furthermore, whenever there is a kill we need to compute the MOE which was defined as timeToFirstKill – timeToFirstDetection. To review, timeToFirstDetection is subtracted to eliminate the effects of the personality factors on the agents' speed of motion.

A closer look reveals that there are two scenarios (S2 and S6) in which no shots and no kills occur. Obviously, these scenarios have input combinations that are capable of avoiding an escalation and the occurrence of violence.

Table 3 provides a "simple stats" overview on the data. Clearly, the combinations of input factors have a strong effect on our MOEs. A look at the first row alone reveals significant differences in the output. Scenarios 2 and 6 (S1, S6) generated no kills at all. In scenarios 1 and 8 (S1, S8), less than 50% of all 100 runs ended with a kill, whereas in Scenarios 3 and 4 (S3, S4) we find killed agents in more than 50% of the cases. Scenarios 5 and 7 (S5, S7) generate kills in every single run.

		S1	S2	S3	S4	S5	S6	S7	S8
Number of runs ending w/ kill		11	0	76	66	100	0	100	45
time from first detection to first kill, given there was a kill	Mean	33.28	#DIV/0!	43.93	78.20	47.37	#DIV/0!	62.43	51.57
	Var	4.75	#DIV/0!	599.28	891.84	13.46	#DIV/0!	121.21	17.66
	Std Dev	2.18	#DIV/0!	24.48	29.86	3.67	#DIV/0!	11.01	4.20
	Lower Quartile	31.61	#NUM!	23.95	58.85	44.61	#NUM!	57.47	49.84
	Min	31.02	0.00	8.43	28.64	41.59	0.00	29.73	43.10
	Median	32.13	#NUM!	44.71	72.36	46.82	#NUM!	64.05	50.86
	Max	36.15	0.00	124.59	135.00	57.18	0.00	90.64	65.14
	Upper Quartile	35.76	#NUM!	62.61	97.20	49.54	#NUM!	67.07	52.93

Table 3. Summary statistics of the output data

Rows two, three and four show there is a huge difference in the mean values, variances and standard deviations between the eight scenarios. Rows five through nine show the data that is necessary to draw boxplot graphs.

Figure 17 shows the boxplots of the output data. Not only are the medians of the MOE different, but also the spread of the data for each scenario. Interestingly, the scenarios seem to be pairwise similar. S2 and S6 don't generate kills at all. S3 and S4 have a huge spread, but are very different in their medians. S5 is similar to S8 in both median and spread, but there also are similarities between S1 and S8. The median of S7 is close to the median of S4; however, S4 has a wider spread.

Of course, we wish to understand the reasons for these similarities. Table 4 provides an overview of the full factorial design for the three variable personality factors. S2 and S6 both have in common a low AffinityToAction (aTa) and a high RiskAversion (rA). With this combination, closeToFriendly (cTf) seems to play an insignificant role.

		Personality Factors		
		CloseToFriendly	AffinityToAction	RiskAversion
Scenario	S1	**L**	**L**	**L**
	S2	**L**	**L**	**H**
	S3	**L**	**H**	**L**
	S4	**L**	**H**	**H**
	S5	**H**	**L**	**L**
	S6	**H**	**L**	**H**
	S7	**H**	**H**	**L**
	S8	**H**	**H**	**H**

Table 4. Experiment design for three input variables

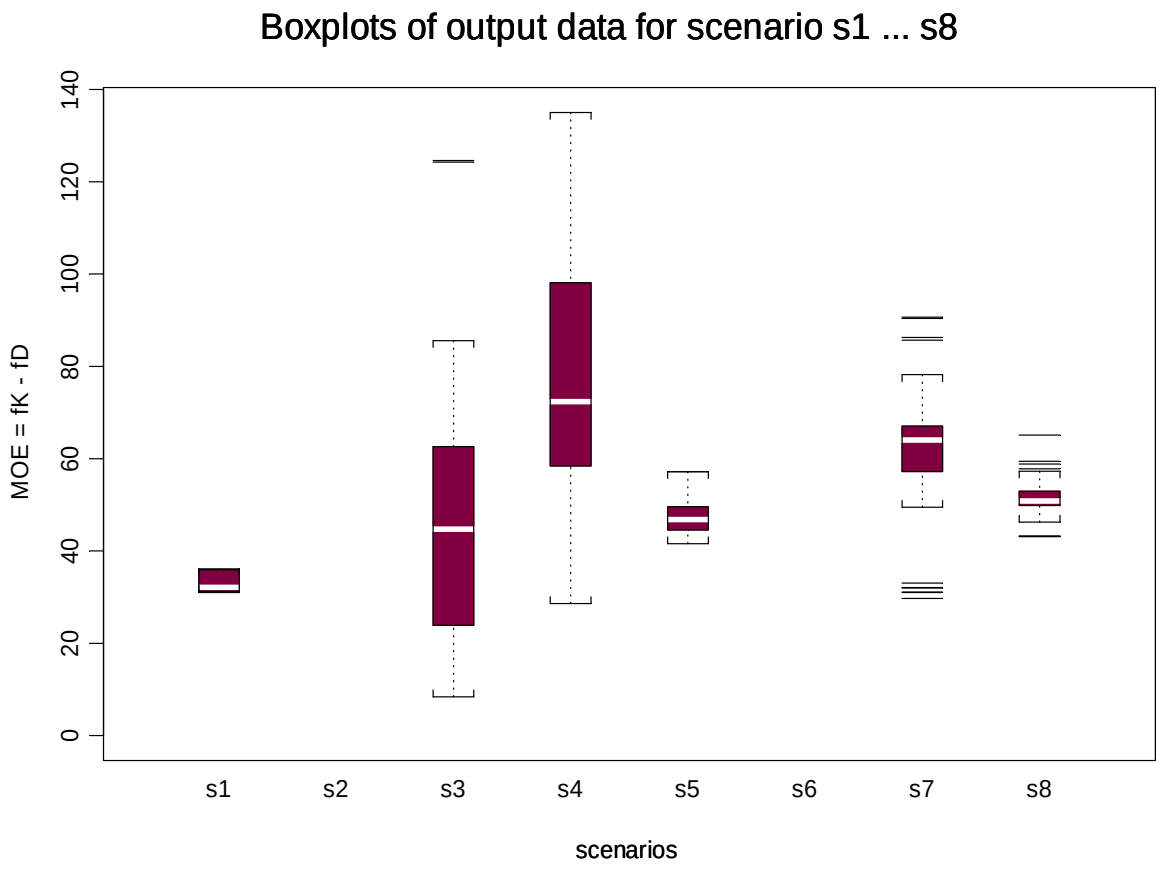


Figure 17. Boxplots graphs of the output data

The huge spread in S3 and S4 seemingly is caused by a low cTf and a high aTa. For this combination, the third variable, rA, causes a remarkable shift in the median MOE. According to one's intuition, if rA gets higher one would expect that the time to the first kill would also increase.

There is no intuitive explanation as to why the outputs of S5 and S8 are so similar. We observe that cTf was held at a high level, whereas both aTa and rA are shifted from low to high. Seemingly, the fact that there is only a minor effect on the spread comes from keeping cTf on a high level. Then, if aTa and rA are equal and changed at the same time there does not seem to be any significant effect on the median MOE.

The similarities between S8 and S1 are intuitive: There is no significant effect when all three variables are changed at the same time. If all variables are low, none is relatively low, and if all are high, none is relatively high.

From S4 to S7, two variables change: cTf is increased and rA is decreased—only aTa remains on a high level. The decrease in spread may be caused by the increase of cTf, but, there is no significant effect on the median by decreasing rA anymore.

To sum up, the combination of a low aTa and a high rA is very unfortunate for our analysis since the situation does not escalate. Watching a single run with such a setting shows that the red agents finally reach their objective, e.g., pass the group of blue peacekeeping agents whose duty it is to keep them from getting there. As a conclusion, the possibility of using force needs to come along with the ability and the attitude to do so. In other words, if a group of peacekeepers fail to use their weapon as an ultimate means they may be unable to keep their opponents from reaching their objective.

Furthermore, it looks like cTf does not affect the MOE as much as the other two variables do. Nonetheless, cTf does have a remarkable effect on the variance. At the moment, we cannot determine which of the two variables aTa and rA have a stronger effect on the MOE, however, there seems to be some interaction between the two.

In the following section two different statistical techniques are applied on the output in order to determine the input variable with the greatest effect and better understand the relationship between our input variables and the MOEs.

2. Classification Tree Categorical Output Data

Our output data can be categorized into two different classes: runs that ended without kills and those that ended with kills. The previous chapter demonstrated that there is a strong influence from the input variables on the occurrence of a kill. As a matter of fact, we have already found out that the combination of a low aTa and a high rA never results in a kill. How about the other combinations?

Classification tree methods are a good choice whenever the purpose is classification. They also allow the prediction of outcomes by a set of if-then rules that are almost self-explaining and can easily be understood. Classification trees are built through a process known as binary recursive partitioning. Recursively, the whole data set is split into partitions in order to split it up further on the branches. The splits occur based on the input variable levels. From among all possible splits, the split that partitions

the node into the two most homogeneous nodes is made. That is, the outcomes in the subnodes are more homogenous than in the parent node.

The whole output data set consists of 800 data points, including 398 points with a kill, i.e., a numeric value for the timeToFirstKill (fK). These data points allow the computation of the MOE (timeToFirstDetection – timeToFirstKill). Consequently, 402 data points do not have a value in the fK column.

In order to build the tree model in S-Plus, the data set needs to be manipulated. A new column with the binary factor (O for no kill, 1 for a kill) needs to be established. S-Plus requires the values in this column to be of type "factor."

Figure 18 displays the classification tree model generated from the complete data set. A classification tree is read top-down. Starting at the root (All Rows), one soon hits the first node. Any split of the line is denoted a node and represents a decision. The decision at the nodes is shown in the headers of the succeeding boxes. In this case, the first split is done asking is aTa = 9 or is aTa = 1? For each setting of the variable, one must follow the respective branch and read the predicted outcome with the according probability in the boxes. For example, at node 1 the data set is split in half. Out of 800 counts, 400 have aTa set to 9. In this subset, the probability of a kill is 0.72. The next split is caused by the variable rA.

Where the lines end, we find the so-called leaves. Leaves also represent potential further nodes. For a discussion of the tree, nodes and leaves are numbered layer-wise (top down, from left to right)

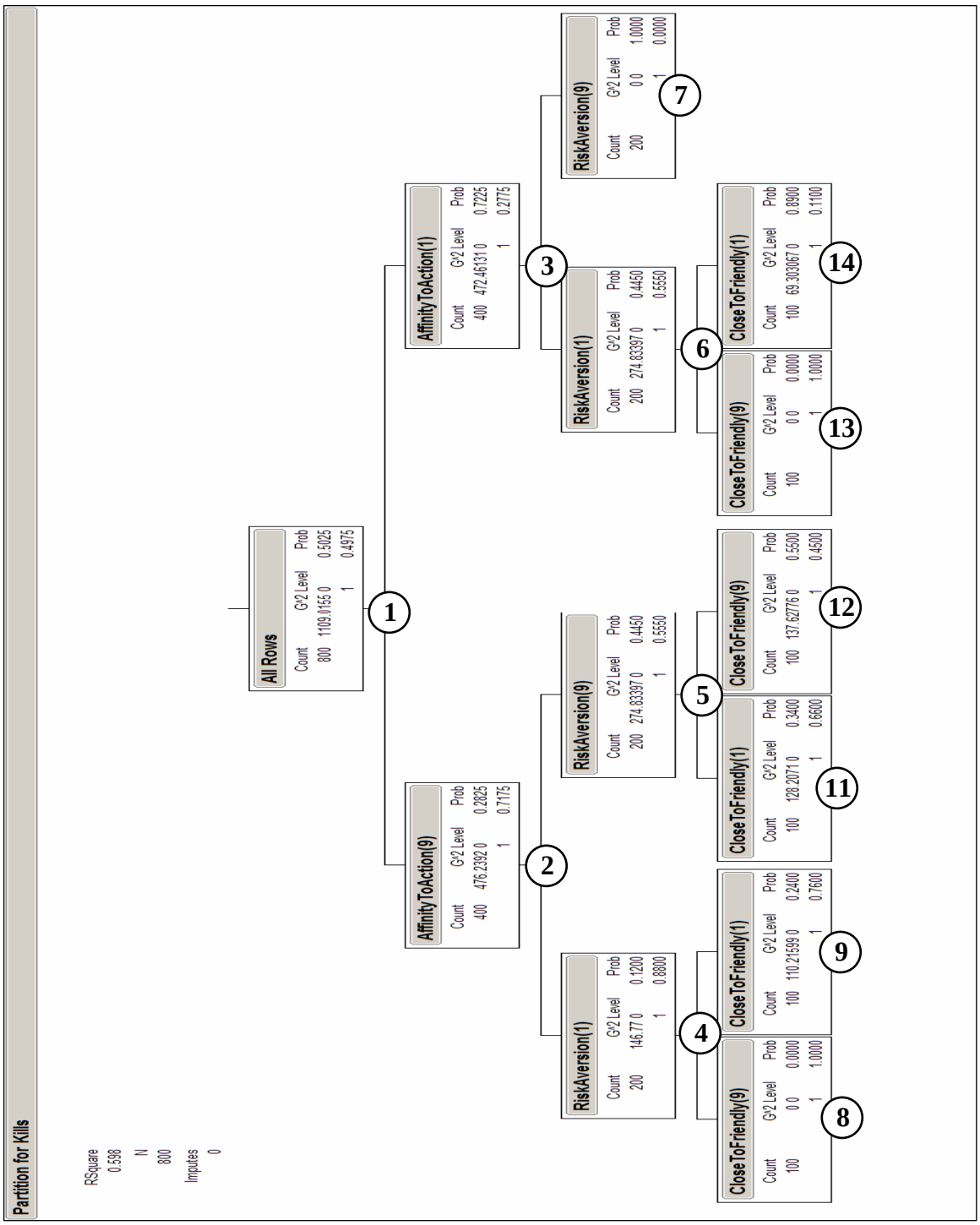


Figure 18. Classification tree model

As it turns out, aTa is the "strongest" variable. It partitions the data set in half. On the second level, we find in both branches rA to be the most influential variable. Theoretically, all three variables could be found here. Therefore, it is clear that the

ranking of the variables is aTa, rA, and then cTf. On the second layer, we also find the first leave. Node 7 shows the case where aTa is low and rA is high. In such cases, the predicted response is 0, meaning no kill. We recall that scenarios S2 and S6 clearly prove this prediction. On this leaf, we have an accuracy of 100%.

On the third level, we can study the effects of the variable cTf. The leaves at nodes number 8 and 9 also have good accuracy. In both cases, aTa is high and rA is low. No matter whether cTf is high or low, the model predicts the outcome to be a kill. These leaves represent scenarios S3 and S7. We find the prediction of node 8 (cTf high), representing scenario S7, to also be 100% accurate. On the other hand, node 9 (cTf low), representing scenario S3, has a misclassification rate of 24%. The tree model predicts all runs to end with a kill, however, only 76 runs actually did.

Furthermore, node 12, representing scenario S5, is very accurate. Out of 100 predicted kills, 100% actually occurred. Node 13, representing scenario S1, turns out to have a misclassification rate of 11%, since the prediction is 0 and the actual kill rate is 11 kills out of 100 runs.

The greatest misclassification rates are found in leaves 10 and 11, which represent scenarios S4 and S8. Both have in common that aTa and rA are set to high. If cTf is low, the tree model predicts a kill with an accuracy of only 66%. If cTf is high, no kill is predicted while 45 kills out of 100 runs actually occurred.

As a summary, Figure 19 provides an overview on the tree model's validity. Out of 800 data points, the model predicts 344 + 342 = 686 correctly. That makes a total misclassification rate of 14%.

```
> table(predict(tree.model, data.logist, type = "class"), data.logist$binary)
    0   1
0 344  56
1  58 342
```

Figure 19. Summary statistics of the classification tree model

3. Regression Model on Measures of Effectiveness (MOEs)

So far, we have analyzed the complete output data. From Table 3, we find that, out of 800 data points, only 398 ended up with a kill. Since we are interested in a mathematical relationship between the input variables and the MOE we need to extract those points that provide a value for the timeToFirstKill. Using this extracted data set of 398 points, a regression model will be built under the premise "given a kill."

Furthermore, the extracted data set, like the one we used for the classification tree model, needs one more column. This column will contain the dependent variable, which is the MOE (timeToFirstKill – timeToFirstDetection). Having prepared the data accordingly, we can use S-Plus to build a regression model. The dependent variable is the MOE, the independent variables are the 398 cases in which the three input variables resulted in a kill (see Table 4).

At first, we start out with a linear model without interaction terms. Figure 20 provides a summary output of the linear regression model without interaction terms. We learn from the P-values that we have statistical significance for the intercept and the variables aTa and rA. However, the P-value of cTf is too high for it to be statistically significant. Also, the R-squared value (0.1489) is small, which means that there are significant deviances between the regression model and the output data.

```
> summary(linear.model)

Call: lm(formula = fK.fD ~ ., data = data.lm)
Residuals:
    Min     1Q Median    3Q   Max
 -44.41 -11.44 0.6379 8.603 71.76

Coefficients:
                  Value Std. Error t value Pr(>|t|)
     (Intercept) 40.2202  3.1909    12.6047  0.0000
 CloseToFriendly  0.3536  0.2749     1.2861  0.1992
AffinityToAction  1.1761  0.3115     3.7757  0.0002
    RiskAversion  1.6777  0.3011     5.5722  0.0000

Residual standard error: 19.65 on 394 degrees of freedom
Multiple R-Squared: 0.1489
F-statistic: 22.98 on 3 and 394 degrees of freedom, the p-value is 1e-013

Correlation of Coefficients:
                 (Intercept) CloseToFriendly AffinityToAction
 CloseToFriendly -0.7498
AffinityToAction -0.7125      0.2937
```

Figure 20. Summary statistics linear regression model on the main effects

Nonetheless, it is worth paying attention to the main effects, e.g., the coefficients to the input variables. Like in the classification tree model, cTf plays a minor role compared to the other two variables. In contrast, rA has a stronger effect on the outcome than aTa. The great surprise, however, is that the coefficients of all three variables are positive. Intuitively, one would think that an increase in aTa would decrease the timeToFirstKill. Of course, these are for only the cases where a kill actually occurred.

In order to improve a linear regression model, S-Plus provides in its Mass library a function called stepAIC. This function adds and drops interaction terms in a stepwise approach in order to better fit the model.

Figure 21 provides the summary output of the linear regression model obtained using stepAIC. Obviously, the stepAIC function added a single interaction term (aTa:rA). We observe that all P-values are very small. Furthermore, R-squared has more than doubled to 0.3443.

```
> summary(linearAIC)

Call: lm(formula = fK.fD ~ CloseToFriendly + AffinityToAction + RiskAversion +
   CloseToFriendly:
   RiskAversion, data = data.lm)
Residuals:
    Min     1Q  Median    3Q  Max
 -49.57 -5.334 -0.3359 4.819 80.2

Coefficients:
                              Value Std. Error  t value Pr(>|t|)
                (Intercept)  21.1282   3.3130     6.3773   0.0000
             CloseToFriendly   2.9022   0.3374     8.6019   0.0000
            AffinityToAction   1.7936   0.2796     6.4145   0.0000
                RiskAversion   4.9180   0.3996    12.3080   0.0000
CloseToFriendly:RiskAversion  -0.6924   0.0640   -10.8223   0.0000

Residual standard error: 17.27 on 393 degrees of freedom
Multiple R-Squared: 0.3443
F-statistic: 51.59 on 4 and 393 degrees of freedom, the p-value is 0

Correlation of Coefficients:
                            (Intercept) CloseToFriendly AffinityToAction RiskAversion
             CloseToFriendly -0.8261
            AffinityToAction -0.6990      0.3483
                RiskAversion -0.4932      0.5935         -0.0558
CloseToFriendly:RiskAversion  0.5325     -0.6980         -0.2041          -0.7493
```

Figure 21. Summary statistics of the linear regression model with interaction term

Looking at the coefficients, we find the intercept to have decreased by a factor of two. All main effects are still positive. The variable rA still has the greatest impact on the outcome. The variable aTa has become the least influential one. Even the variable cTf has a greater effect on the timeToFirstKill.

The new term that caused the improvements of the P-values and R-squared is the interaction term. Its coefficient is negative, but it is significantly smaller than the two main effects. Thus, increasing either aTa or rA results in a limited increase in the outcome.

To sum up, the linear regression model is anything but intuitive. Given that there is a kill, the smaller all three variables are, the sooner the kill occurs. For rA, the positive sign and the largest value of the coefficient is no big surprise. One would think that increasing the rA of all agents will delay the time until the first kill. Likewise, for cTf: if all agents stay closely in their group, shootings may occur later because panic reactions, as one of the reasons for shootings, are limited. The big surprise, however, is that, given there is a kill, the greater the agent's aTa is, the later it is going to occur. No intuitive explanation to this fact has been found.

At this point, the author points out one more time that this result may be a result of the fact that the regression model is built on a subset of the whole output data. We use only 398 out of 800 data points that we generated to build this model, since we can only use those points where a kill actually occurred. Furthermore, certain combinations of the input variables are not represented in the regression model. The combinations of scenarios S2 and S6 (aTa low, rA high) did not add a single data point to the model. The reader may recall that both scenarios ended up with no kills at all, not even a shot, in 100 runs each. Finally, the settings of the input factors were relatively extreme (1.0 for the low and 9.0 for the high setting) with respect to the range of the model's variables (from 0.0 through 10.0). It may be interesting for future research to revisit this model again using input settings that are not so extreme.

Further research, particularly an analysis of all personality factors, is necessary to gain more insight into this phenomenon.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The purpose of this thesis was to create an upgradeable agent-based simulation tool that allows research on specific situations of peacekeeping missions. The architecture of the computer program was to incorporate a high level of upgradeability. ABS and DES, as two different simulation concepts, should be brought together and combined in a simulation model. Furthermore, it was required to make the model data-farmable. Nonetheless, an optional animation that allows the study of the agents' behavior in single runs was desirable. With respect to the random number management, the model was required to allow multi-runs, many replications, and the results need to be reproducible.

As it turns out, Simkit is capable of serving as a basis for ABS. Building an ABS on a Simkit basis provides a series of advantages. Using Simkit, the model will automatically follow a discrete-event approach. Listener patterns and loosely coupled components guarantee that the access to information (of other agents) is limited to a degree that represents the real world. The use of Factories, among other Simkit approaches, makes the simulation model stable and robust. An animation that allows the observation of the agents' motions is added by Simkit for free.

Chapter III of this thesis provides an exemplary analysis. Above all, this part is meant to demonstrate that the model actually is data-farmable. Out of the almost unlimited amount of different input combinations, an analysis of personality factors was carried out. Out of the seven personality factors (including the training level) that characterize an agent, three were picked as input variables: CloseToFriendly, AffinityToAction, and RiskAversion. A two-level full factorial design was chosen and each combination of input factors was run 100 times. As for the output, the primary MOE was defined as timeToFirstKill – timeToFirstDetection. By subtracting the second term from the first, any effects of the input factors on the speed of motion of the agents is eliminated. Therefore, we really consider the time of the escalation of the situation. The

stopwatch is started as soon as the first blue agent detects a red agent or vice versa and is clicked when the first agent gets killed.

The model was able to run each of the scenarios, represented in a single XML input file, successively. It carried out 100 runs on each scenario and created a text file for the output of each scenario. It turned out that on an average home PC (Pentium 4, 2.66 GHz, 512 MB) the execution of 100 runs of such a scenario (8 blue agents, 24 red agents) lasted about 30 minutes. Further research is necessary to gain more experience in the computational effort when the number of agents is increased.

A first look at the output data revealed that a kill does not occur in every run. In fact, there are two scenarios (e.g., two different input combinations) that did not generate a kill at all. Therefore, it was impossible to apply linear regression to the complete output data for the time to first kill measure.

Applying classification tree techniques allowed answering the question: Which one of the three input factors has the greatest effect on the occurrence of kills? As it turned out in this scenario, it is AffintyToAction. The second most influential variable is RiskAversion, followed by CloseToFriendly. Furthermore, the tree model also brought forth that the combination AffinityToAction low and RiskAversion high results in no kills at all. This suggestion of the model is in absolute accordance with the output data. Strictly speaking this would be a very desirable situation for peacekeeping missions. One could think that through the military training and preparations for a peacekeeping mission, the soldiers should take on a very passive and risk-averting attitude and everything would be perfect. The answer to this was given by the model itself. Taking advantage of the animation and watching one of the runs with this setting showed that the red agents finally broke through and successfully made their way to their objective. Of course, in some peacekeeping missions, this tactical failure may be better than a kill, which could, in a strategic sense, result in overall mission failure.

On the other hand, if RiskAversion is low and AffinityToAction is high, no matter what the setting of CloseToFriendly is, the model always suggests a kill. Compared to the output data we have a misclassification rate of 12% in this situation. Anyway, the attitude of soldiers represented by this input setting is definitely unfortunate for

peacekeeping missions. The group of peacekeepers may be successful in keeping the crowd of demonstrators from reaching their objective; however, only at the price of people being killed.

The last portion of the analysis part dealt with a subset of the whole output data set. In order to build a linear regression model, only those data points that included a timeToFirstKill were useful. The question here was: Given a kill, what is the influence of the input factors on the MOE? The linear regression model suggests that RiskAversion is the most influential input factor, followed by CloseToFriendly and AffinityToAction. All factors are positive, meaning that increasing each one of them results in a greater time to first kill. In other words, the model suggests that, given a kill, increasing the AffinityToAction of all agents' results in a situation that escalates slower. This is a surprise. There is no intuitive explanation to this finding. Further research needs to be undertaken in order to gain more insight into this phenomenon.

Such surprises that cannot be immediately explained are not new in the world of ABS. As a matter of fact, the purpose of ABS is to generate questions rather than answers. John Holland, a founder of CAS theory, said:

> I just love these things where the situation unfolds and I say, 'Gee whiz! Did that really come from these assumptions!?' Because if I do it right, if the underlying rules of evolution of the themes are in control and not me, then I'll be surprised. And if I'm not surprised, then I am not very happy, because I know I've built everything in from the start. [Waldrop, 1992]

B. RECOMMENDATIONS FOR FURTHER RESEARCH

Throughout the text, we have seen several suggestions and recommendations for further work. This chapter provides a summary on these recommendations, as this thesis could be extended in two different directions:

1) Upgrades and improvements to the simulation model.

2) Further analysis using the existing model.

Both ways can be explored independently of one another; however, it is also possible to carry out further research with a combination of both. For the first way, thorough programming skills and a good understanding of DES using Simkit is

necessary. The second way also requires programming skills, but only to a very basic level. The main requirement here is a concern for statistical techniques and some knowledge in state-of-the-art designs of experiment.

1. Upgrades on the Model

Upgradeability was a key design requirement. This means that the model is capable of dealing with future PKO challenges. The upgradeable structure of the software is described in detail in Section II.B.4. Recommendations for further upgrades are given in Sections II.C and II.D. As there is no limit to one's creativity, this paragraph provides a brief list of further upgrades that the author would recommend starting with.

As it comes to an analysis of specific situations, the model's animation is limited to the movement processes. Shots are not yet displayed. Killed and wounded agents cannot be differentiated from healthy agents. It may be desirable to upgrade the animation at first. Since the animation used in this thesis is purely Simkit-based, the author recommends getting in touch with Professor Buss, NPS, as soon as an upgrade to the existing animation is intended.

Currently, the *AgentSensor* is based on the cookie cutter principle. A cookie cutter sensor detects all other agents with range with 100% accuracy. It may be interesting to test various other sensing mechanisms (and their effects on the outcome). A good start may be to add probabilistic sensing. The probability of detection may be high when the other agent is in front and low if it is behind the agent.

Furthermore, fuzziness could be added to the information flow within an agent. As shown in Figure 6 (Layer model of the *BasicAgent*) any piece of information that is processed in the *Agent* level must be passed through the *InnerEnvironment* level. Instead of just passing it through, randomness could be added.

The *BasicAgent* is already prepared to incorporate a *History*. It could be interesting to introduce a history mechanism. For example, whenever an agent has already tried to take a certain action to reach a goal, yet was not successful, a potential History implementation could tell him that he should try to take a different action. In this respect, an extended set of tickets (actions) could be associated to each goal.

In order to study scenarios in urban terrain better, buildings and other sorts of infrastructure may be introduced. Care must be taken here: in order to model infrastructure adequately, the whole sensing mechanism must be equipped with a line-of-sight algorithm. Such algorithms usually consume a huge amount of computing power, i.e., they slow the simulation down. Furthermore, the author would recommend introducing a PathMoverManager[26] for the agents, so that they can nicely move around buildings, cross bridges, etc.

For potential analysis of the effects of leadership, a mechanism of giving and receiving orders could be introduced. As explained in Section II.C.3, adding doGiveOrder(Order) and doReceiveOrder(Order) methods to the agents would be a start to this enhancement.

Last, but not least, the algorithm that evaluates hits, wounds and kills may be modeled in a more sophisticated way. Currently, random number draws determine the effects of shots. One approach would be a 2-D shape of a person on an underlying coordinate system. Hits in predefined areas of this shape either lead to a miss, a wound, or a kill event.

The reader may get an impression that there is no limit to one's creativity. However, an ABM may not necessarily become better by continuously refining it. The complexity of the aggregated behavior of an ABM does not depend on highly complex agent designs. From a certain degree on, the complexity of the system as a whole, as well as its potential outcome, may even decrease as the fidelity of the agents is further increased.

The model, as it is, is capable of simulating a PKO scenario as the situation escalates in a reasonable way. Without any further upgrades on the software, there is a variety of further research possible.

[26] Simkit already provides a variety of PathMoverManagers.

2. Recommendations on Further Analysis

The exemplary analysis provided in Chapter III of this thesis was meant to prove that the model is data-farmable and to give some insight into the relationship of the personality factors and the outcome of the simulation.

A possible starting point to further research could be an analysis of all personality factors (including training). For a thorough exploration, it is highly recommended to not limit the settings of these input factors to only two levels. However, a three-level, full factorial design would mean $3^7 = 2187$ scenarios. Therefore, further statistical research needs to include modern experimental designs, such as Latin Hypercubes.

It may also be interesting to study the effect of variable force sizes. This thesis was done on a single PC. If high performance computing power is available, such as the Maui High Performance Computing Center (MHPCC) provides, an exploration of variable force ratios may generate surprising insights.

While the exemplary analysis is based on the assumption of homogeneous groups of agents, the author would be interested in the effect of heterogeneous groups of agents. The simulation model already includes a mechanism to generate personality factors that are based on a normal distribution.

Furthermore, the model could serve as a tool to explore different sorts of tactics. For example, given a group of demonstrators, the peacekeepers may encounter them in a single group or split into two or more smaller groups.

So far, the reader may have an idea of a possible first extension to the existing analysis. Again, there is no limitation to one's creativity. As time allows, the author is more than happy to provide assistance to those who are willing to exploit this model. My contact information can be found in Appendix C of this thesis.

APPENDIX A. GERMAN TO ENGLISH REFERENCE

Bundeswehr	German Federal Armed Forces
Verteidigungspolitiche Richtlinien	Defence Policy Guidelines
Konzeption der Bundeswehr	Bundeswehr Concept
Nationale Volksarmee	National People's Army

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. SIMULATION OUTPUT

inputFileName[i] = ../scenarios/analysis_ff_01			
fD	fS	fK	
16.529	50.306		>>>>> run number 1 is done!!! <<<<<
20.834	50.26		>>>>> run number 2 is done!!! <<<<<
20.788	47.196		>>>>> run number 3 is done!!! <<<<<
20.573	46.318		>>>>> run number 4 is done!!! <<<<<
19.960	50.207		>>>>> run number 5 is done!!! <<<<<
20.741	49.957		>>>>> run number 6 is done!!! <<<<<
20.839	50.056		>>>>> run number 7 is done!!! <<<<<
20.552	50.839		>>>>> run number 8 is done!!! <<<<<
21.169	51.491		>>>>> run number 9 is done!!! <<<<<
20.663	46.831		>>>>> run number 10 is done!!! <<<<<
20.001	50.702		>>>>> run number 11 is done!!! <<<<<
21.245	50.494		>>>>> run number 12 is done!!! <<<<<
20.353	46.737		>>>>> run number 13 is done!!! <<<<<
20.519	46.462		>>>>> run number 14 is done!!! <<<<<
20.003	50.488		>>>>> run number 15 is done!!! <<<<<
19.132	49.705		>>>>> run number 16 is done!!! <<<<<
20.864	50.437	52.125	>>>>> run number 17 is done!!! <<<<<
19.979	50.511		>>>>> run number 18 is done!!! <<<<<
20.862	51.413	53.65	>>>>> run number 19 is done!!! <<<<<
16.088	46.617		>>>>> run number 20 is done!!! <<<<<
15.865	50.485		>>>>> run number 21 is done!!! <<<<<
16.646	50.885	52.797	>>>>> run number 22 is done!!! <<<<<
20.550	50.478		>>>>> run number 23 is done!!! <<<<<
20.534	50.891		>>>>> run number 24 is done!!! <<<<<
20.890	46.456		>>>>> run number 25 is done!!! <<<<<
20.805	50.613		>>>>> run number 26 is done!!! <<<<<
20.282	50.406		>>>>> run number 27 is done!!! <<<<<
20.016	50.47		>>>>> run number 28 is done!!! <<<<<
20.001	50.231		>>>>> run number 29 is done!!! <<<<<
16.039	50.37		>>>>> run number 30 is done!!! <<<<<
20.945	50.536		>>>>> run number 31 is done!!! <<<<<
20.931	47.078		>>>>> run number 32 is done!!! <<<<<
20.791	50.288	51.882	>>>>> run number 33 is done!!! <<<<<
20.343	50.447	52.397	>>>>> run number 34 is done!!! <<<<<
19.912	50.595		>>>>> run number 35 is done!!! <<<<<
20.886	50.427		>>>>> run number 36 is done!!! <<<<<
20.595	46.79		>>>>> run number 37 is done!!! <<<<<
18.562	51.471		>>>>> run number 38 is done!!! <<<<<
20.077	50.434	52.036	>>>>> run number 39 is done!!! <<<<<
20.768	50.381		>>>>> run number 40 is done!!! <<<<<
20.435	46.719		>>>>> run number 41 is done!!! <<<<<
20.771	46.929		>>>>> run number 42 is done!!! <<<<<
21.008	46.502		>>>>> run number 43 is done!!! <<<<<
20.798	47.003		>>>>> run number 44 is done!!! <<<<<
20.783	50.752		>>>>> run number 45 is done!!! <<<<<
20.787	50.521	52.921	>>>>> run number 46 is done!!! <<<<<
16.998	44.66		>>>>> run number 47 is done!!! <<<<<
19.139	50.812		>>>>> run number 48 is done!!! <<<<<

16.859	50.277		>>>>> run number 49 is done!!! <<<<<
20.017	50.508		>>>>> run number 50 is done!!! <<<<<
19.876	50.151		>>>>> run number 51 is done!!! <<<<<
20.042	50.282		>>>>> run number 52 is done!!! <<<<<
20.877	49.804		>>>>> run number 53 is done!!! <<<<<
16.568	50.796	52.692	>>>>> run number 54 is done!!! <<<<<
20.916	50.423		>>>>> run number 55 is done!!! <<<<<
20.026	50.332		>>>>> run number 56 is done!!! <<<<<
20.385	49.817		>>>>> run number 57 is done!!! <<<<<
20.776	50.114	51.792	>>>>> run number 58 is done!!! <<<<<
15.996	50.973		>>>>> run number 59 is done!!! <<<<<
20.884	47.088		>>>>> run number 60 is done!!! <<<<<
20.675	43.273		>>>>> run number 61 is done!!! <<<<<
16.074	50.702		>>>>> run number 62 is done!!! <<<<<
20.833	49.949		>>>>> run number 63 is done!!! <<<<<
16.598	50.615	52.56	>>>>> run number 64 is done!!! <<<<<
20.751	46.697		>>>>> run number 65 is done!!! <<<<<
20.815	46.588		>>>>> run number 66 is done!!! <<<<<
19.134	51.041		>>>>> run number 67 is done!!! <<<<<
20.317	46.274		>>>>> run number 68 is done!!! <<<<<
20.486	44.186		>>>>> run number 69 is done!!! <<<<<
20.806	50.654		>>>>> run number 70 is done!!! <<<<<
20.326	51.142		>>>>> run number 71 is done!!! <<<<<
16.839	46.901		>>>>> run number 72 is done!!! <<<<<
20.740	50.637		>>>>> run number 73 is done!!! <<<<<
16.571	50.398		>>>>> run number 74 is done!!! <<<<<
19.949	50.409		>>>>> run number 75 is done!!! <<<<<
20.784	50.693		>>>>> run number 76 is done!!! <<<<<
20.977	50.237		>>>>> run number 77 is done!!! <<<<<
16.146	52.085		>>>>> run number 78 is done!!! <<<<<
20.928	44.41		>>>>> run number 79 is done!!! <<<<<
20.325	49.949		>>>>> run number 80 is done!!! <<<<<
20.020	50.445		>>>>> run number 81 is done!!! <<<<<
19.112	46.935		>>>>> run number 82 is done!!! <<<<<
16.592	50.278		>>>>> run number 83 is done!!! <<<<<
20.925	50.966		>>>>> run number 84 is done!!! <<<<<
20.921	49.901		>>>>> run number 85 is done!!! <<<<<
20.604	50.463		>>>>> run number 86 is done!!! <<<<<
20.748	50.361		>>>>> run number 87 is done!!! <<<<<
18.766	50.848		>>>>> run number 88 is done!!! <<<<<
20.893	50.454		>>>>> run number 89 is done!!! <<<<<
20.736	49.665		>>>>> run number 90 is done!!! <<<<<
16.841	50.331	52.407	>>>>> run number 91 is done!!! <<<<<
20.905	51.199		>>>>> run number 92 is done!!! <<<<<
17.025	50.054		>>>>> run number 93 is done!!! <<<<<
22.869	50.284		>>>>> run number 94 is done!!! <<<<<
20.832	46.86		>>>>> run number 95 is done!!! <<<<<
20.783	44.311		>>>>> run number 96 is done!!! <<<<<
16.032	46.463		>>>>> run number 97 is done!!! <<<<<
20.729	46.862		>>>>> run number 98 is done!!! <<<<<
16.088	50.287		>>>>> run number 99 is done!!! <<<<<
20.013	50.118		>>>>> run number 100 is done!!! <<<<<

inputFileName[i] = ../scenarios/analysis_ff_02			
fD			
11.539			>>>>> run number 1 is done!!! <<<<<
12.019			>>>>> run number 2 is done!!! <<<<<
11.665			>>>>> run number 3 is done!!! <<<<<
11.733			>>>>> run number 4 is done!!! <<<<<
11.729			>>>>> run number 5 is done!!! <<<<<
11.774			>>>>> run number 6 is done!!! <<<<<
11.435			>>>>> run number 7 is done!!! <<<<<
11.535			>>>>> run number 8 is done!!! <<<<<
11.537			>>>>> run number 9 is done!!! <<<<<
11.610			>>>>> run number 10 is done!!! <<<<<
11.552			>>>>> run number 11 is done!!! <<<<<
11.506			>>>>> run number 12 is done!!! <<<<<
11.471			>>>>> run number 13 is done!!! <<<<<
11.931			>>>>> run number 14 is done!!! <<<<<
11.597			>>>>> run number 15 is done!!! <<<<<
11.652			>>>>> run number 16 is done!!! <<<<<
11.730			>>>>> run number 17 is done!!! <<<<<
11.681			>>>>> run number 18 is done!!! <<<<<
11.491			>>>>> run number 19 is done!!! <<<<<
11.432			>>>>> run number 20 is done!!! <<<<<
11.477			>>>>> run number 21 is done!!! <<<<<
11.484			>>>>> run number 22 is done!!! <<<<<
11.468			>>>>> run number 23 is done!!! <<<<<
11.665			>>>>> run number 24 is done!!! <<<<<
11.445			>>>>> run number 25 is done!!! <<<<<
11.538			>>>>> run number 26 is done!!! <<<<<
11.505			>>>>> run number 27 is done!!! <<<<<
11.859			>>>>> run number 28 is done!!! <<<<<
11.496			>>>>> run number 29 is done!!! <<<<<
11.626			>>>>> run number 30 is done!!! <<<<<
11.896			>>>>> run number 31 is done!!! <<<<<
11.694			>>>>> run number 32 is done!!! <<<<<
11.599			>>>>> run number 33 is done!!! <<<<<
11.432			>>>>> run number 34 is done!!! <<<<<
11.606			>>>>> run number 35 is done!!! <<<<<
11.896			>>>>> run number 36 is done!!! <<<<<
11.658			>>>>> run number 37 is done!!! <<<<<
11.527			>>>>> run number 38 is done!!! <<<<<
11.725			>>>>> run number 39 is done!!! <<<<<
11.572			>>>>> run number 40 is done!!! <<<<<
11.717			>>>>> run number 41 is done!!! <<<<<
11.728			>>>>> run number 42 is done!!! <<<<<
11.559			>>>>> run number 43 is done!!! <<<<<
11.940			>>>>> run number 44 is done!!! <<<<<
11.427			>>>>> run number 45 is done!!! <<<<<
11.720			>>>>> run number 46 is done!!! <<<<<
11.643			>>>>> run number 47 is done!!! <<<<<
11.908			>>>>> run number 48 is done!!! <<<<<
11.991			>>>>> run number 49 is done!!! <<<<<
12.203			>>>>> run number 50 is done!!! <<<<<
11.822			>>>>> run number 51 is done!!! <<<<<

11.854			>>>>> run number 52 is done!!! <<<<<
11.442			>>>>> run number 53 is done!!! <<<<<
11.726			>>>>> run number 54 is done!!! <<<<<
11.552			>>>>> run number 55 is done!!! <<<<<
11.566			>>>>> run number 56 is done!!! <<<<<
11.581			>>>>> run number 57 is done!!! <<<<<
11.503			>>>>> run number 58 is done!!! <<<<<
11.523			>>>>> run number 59 is done!!! <<<<<
11.456			>>>>> run number 60 is done!!! <<<<<
11.532			>>>>> run number 61 is done!!! <<<<<
11.469			>>>>> run number 62 is done!!! <<<<<
11.627			>>>>> run number 63 is done!!! <<<<<
11.537			>>>>> run number 64 is done!!! <<<<<
11.812			>>>>> run number 65 is done!!! <<<<<
11.516			>>>>> run number 66 is done!!! <<<<<
11.808			>>>>> run number 67 is done!!! <<<<<
11.466			>>>>> run number 68 is done!!! <<<<<
11.653			>>>>> run number 69 is done!!! <<<<<
11.614			>>>>> run number 70 is done!!! <<<<<
11.648			>>>>> run number 71 is done!!! <<<<<
11.500			>>>>> run number 72 is done!!! <<<<<
11.776			>>>>> run number 73 is done!!! <<<<<
11.454			>>>>> run number 74 is done!!! <<<<<
11.527			>>>>> run number 75 is done!!! <<<<<
11.521			>>>>> run number 76 is done!!! <<<<<
11.954			>>>>> run number 77 is done!!! <<<<<
11.475			>>>>> run number 78 is done!!! <<<<<
11.428			>>>>> run number 79 is done!!! <<<<<
11.424			>>>>> run number 80 is done!!! <<<<<
11.502			>>>>> run number 81 is done!!! <<<<<
11.662			>>>>> run number 82 is done!!! <<<<<
11.487			>>>>> run number 83 is done!!! <<<<<
11.454			>>>>> run number 84 is done!!! <<<<<
11.604			>>>>> run number 85 is done!!! <<<<<
11.745			>>>>> run number 86 is done!!! <<<<<
11.471			>>>>> run number 87 is done!!! <<<<<
11.595			>>>>> run number 88 is done!!! <<<<<
11.632			>>>>> run number 89 is done!!! <<<<<
11.664			>>>>> run number 90 is done!!! <<<<<
11.589			>>>>> run number 91 is done!!! <<<<<
11.455			>>>>> run number 92 is done!!! <<<<<
11.808			>>>>> run number 93 is done!!! <<<<<
11.523			>>>>> run number 94 is done!!! <<<<<
11.970			>>>>> run number 95 is done!!! <<<<<
11.582			>>>>> run number 96 is done!!! <<<<<
11.595			>>>>> run number 97 is done!!! <<<<<
11.676			>>>>> run number 98 is done!!! <<<<<
11.594			>>>>> run number 99 is done!!! <<<<<
11.520			>>>>> run number 100 is done!!! <<<<<

inputFileName[i] = ../scenarios/analysis_ff_03			
fD	fS	fK	

15.746	23.377	70.406	>>>>> run number 1 is done!!! <<<<<
15.633	23.275		>>>>> run number 2 is done!!! <<<<<
15.763	23.05	79.56	>>>>> run number 3 is done!!! <<<<<
15.660	22.994		>>>>> run number 4 is done!!! <<<<<
15.669	23.585	37.906	>>>>> run number 5 is done!!! <<<<<
15.803	22.913		>>>>> run number 6 is done!!! <<<<<
15.748	23.531		>>>>> run number 7 is done!!! <<<<<
15.631	23.283	81.403	>>>>> run number 8 is done!!! <<<<<
15.633	23.666		>>>>> run number 9 is done!!! <<<<<
15.635	23.547	101.22	>>>>> run number 10 is done!!! <<<<<
15.641	23.392	140.235	>>>>> run number 11 is done!!! <<<<<
15.742	23.205	88.196	>>>>> run number 12 is done!!! <<<<<
15.716	23.557	53.473	>>>>> run number 13 is done!!! <<<<<
15.729	23.019	26.157	>>>>> run number 14 is done!!! <<<<<
15.653	23.487	82.418	>>>>> run number 15 is done!!! <<<<<
15.982	22.999		>>>>> run number 16 is done!!! <<<<<
15.876	23.377		>>>>> run number 17 is done!!! <<<<<
15.787	23.349	68.692	>>>>> run number 18 is done!!! <<<<<
15.678	23.286		>>>>> run number 19 is done!!! <<<<<
15.687	23.268	51.882	>>>>> run number 20 is done!!! <<<<<
15.671	23.508	65.834	>>>>> run number 21 is done!!! <<<<<
15.689	23.343	25.502	>>>>> run number 22 is done!!! <<<<<
15.744	23.227	64.472	>>>>> run number 23 is done!!! <<<<<
15.659	23.571	52.219	>>>>> run number 24 is done!!! <<<<<
15.920	23.226	79.45	>>>>> run number 25 is done!!! <<<<<
15.646	23.564	67.094	>>>>> run number 26 is done!!! <<<<<
15.646	23.578	61.121	>>>>> run number 27 is done!!! <<<<<
15.630	22.872	78.229	>>>>> run number 28 is done!!! <<<<<
15.649	23.101	57.35	>>>>> run number 29 is done!!! <<<<<
15.656	23.423	58.523	>>>>> run number 30 is done!!! <<<<<
15.657	23.708	35.076	>>>>> run number 31 is done!!! <<<<<
15.703	23.004	82.533	>>>>> run number 32 is done!!! <<<<<
15.671	23.514		>>>>> run number 33 is done!!! <<<<<
15.662	23.244		>>>>> run number 34 is done!!! <<<<<
15.679	23.287	56.403	>>>>> run number 35 is done!!! <<<<<
15.653	23.445	27.049	>>>>> run number 36 is done!!! <<<<<
15.657	23.565	76.593	>>>>> run number 37 is done!!! <<<<<
15.791	22.916	24.221	>>>>> run number 38 is done!!! <<<<<
15.666	23.206	54.154	>>>>> run number 39 is done!!! <<<<<
15.647	23.454	36.478	>>>>> run number 40 is done!!! <<<<<
15.659	23.109	72.276	>>>>> run number 41 is done!!! <<<<<
15.692	23.005	62.989	>>>>> run number 42 is done!!! <<<<<
15.675	22.935		>>>>> run number 43 is done!!! <<<<<
15.749	23.179		>>>>> run number 44 is done!!! <<<<<
15.684	23.459		>>>>> run number 45 is done!!! <<<<<
15.756	23.539	41.183	>>>>> run number 46 is done!!! <<<<<
15.676	23.527	62.67	>>>>> run number 47 is done!!! <<<<<
15.640	22.975	64.018	>>>>> run number 48 is done!!! <<<<<
15.730	23.224	83.217	>>>>> run number 49 is done!!! <<<<<
15.667	23.176	25.208	>>>>> run number 50 is done!!! <<<<<
15.664	23.525		>>>>> run number 51 is done!!! <<<<<
15.731	23.373	76.198	>>>>> run number 52 is done!!! <<<<<
15.644	23.455	82.396	>>>>> run number 53 is done!!! <<<<<
15.686	23.284	24.798	>>>>> run number 54 is done!!! <<<<<

15.648	23.299	74.575	>>>>> run number 55 is done!!! <<<<<
15.689	23.508	25.356	>>>>> run number 56 is done!!! <<<<<
15.631	23.354	27.12	>>>>> run number 57 is done!!! <<<<<
15.647	23.243	76.473	>>>>> run number 58 is done!!! <<<<<
15.636	23.072		>>>>> run number 59 is done!!! <<<<<
15.694	23.14	39.704	>>>>> run number 60 is done!!! <<<<<
15.938	23.102	71.731	>>>>> run number 61 is done!!! <<<<<
15.757	23.566		>>>>> run number 62 is done!!! <<<<<
15.763	23.538		>>>>> run number 63 is done!!! <<<<<
15.644	23.507	84.08	>>>>> run number 64 is done!!! <<<<<
15.689	23.519	52.867	>>>>> run number 65 is done!!! <<<<<
15.670	23.494	82.163	>>>>> run number 66 is done!!! <<<<<
15.633	23.071	79.771	>>>>> run number 67 is done!!! <<<<<
15.693	22.939	39.473	>>>>> run number 68 is done!!! <<<<<
15.638	23.548	53.736	>>>>> run number 69 is done!!! <<<<<
15.714	23.331	43.166	>>>>> run number 70 is done!!! <<<<<
15.639	23.31		>>>>> run number 71 is done!!! <<<<<
15.775	23.523		>>>>> run number 72 is done!!! <<<<<
15.674	23.395		>>>>> run number 73 is done!!! <<<<<
15.629	23.109		>>>>> run number 74 is done!!! <<<<<
15.669	23.259	59.608	>>>>> run number 75 is done!!! <<<<<
15.642	23.668	33.23	>>>>> run number 76 is done!!! <<<<<
15.723	23.143	33.834	>>>>> run number 77 is done!!! <<<<<
15.653	22.829	88.522	>>>>> run number 78 is done!!! <<<<<
15.710	22.938	26.129	>>>>> run number 79 is done!!! <<<<<
15.718	23.32	24.964	>>>>> run number 80 is done!!! <<<<<
15.698	23.405	51.231	>>>>> run number 81 is done!!! <<<<<
15.688	23.507		>>>>> run number 82 is done!!! <<<<<
15.685	23.519	38.241	>>>>> run number 83 is done!!! <<<<<
15.694	23.542	37.982	>>>>> run number 84 is done!!! <<<<<
15.657	23.193		>>>>> run number 85 is done!!! <<<<<
15.664	23.62	69.927	>>>>> run number 86 is done!!! <<<<<
15.712	22.99	139.969	>>>>> run number 87 is done!!! <<<<<
15.741	23.591		>>>>> run number 88 is done!!! <<<<<
15.744	23.141	25.156	>>>>> run number 89 is done!!! <<<<<
15.747	22.972	42.56	>>>>> run number 90 is done!!! <<<<<
15.668	23.098	78.298	>>>>> run number 91 is done!!! <<<<<
15.722	23.224	81.03	>>>>> run number 92 is done!!! <<<<<
15.633	23.528	79.684	>>>>> run number 93 is done!!! <<<<<
15.787	23.139	41.657	>>>>> run number 94 is done!!! <<<<<
15.635	23.585	71.425	>>>>> run number 95 is done!!! <<<<<
15.784	22.932	47.561	>>>>> run number 96 is done!!! <<<<<
15.715	23.496	67.522	>>>>> run number 97 is done!!! <<<<<
15.658	23.514	78.522	>>>>> run number 98 is done!!! <<<<<
15.753	23.373	41.56	>>>>> run number 99 is done!!! <<<<<
15.650	23.609	43.75	>>>>> run number 100 is done!!! <<<<<

inputFileName[i] = ../scenarios/analysis_ff_04			
fD	fS	fK	
11.172	26.554	84.933	>>>>> run number 1 is done!!! <<<<<
11.414	26.196	134.321	>>>>> run number 2 is done!!! <<<<<
11.186	26.16	130.852	>>>>> run number 3 is done!!! <<<<<

11.156	26.141	105.557	>>>>> run number 4 is done!!! <<<<<
11.232	25.922		>>>>> run number 5 is done!!! <<<<<
11.630	26.266		>>>>> run number 6 is done!!! <<<<<
11.145	26.37	77.681	>>>>> run number 7 is done!!! <<<<<
11.305	26.462		>>>>> run number 8 is done!!! <<<<<
11.195	26.53	71.278	>>>>> run number 9 is done!!! <<<<<
11.286	26.287		>>>>> run number 10 is done!!! <<<<<
11.142	26.221	131.373	>>>>> run number 11 is done!!! <<<<<
11.145	26.274	81.286	>>>>> run number 12 is done!!! <<<<<
11.169	26.512	93.411	>>>>> run number 13 is done!!! <<<<<
11.130	26.556		>>>>> run number 14 is done!!! <<<<<
11.127	26.039	64.352	>>>>> run number 15 is done!!! <<<<<
11.162	26.253	84.618	>>>>> run number 16 is done!!! <<<<<
11.156	26.372	87.463	>>>>> run number 17 is done!!! <<<<<
11.122	25.969	43.405	>>>>> run number 18 is done!!! <<<<<
11.208	26.554		>>>>> run number 19 is done!!! <<<<<
11.350	26.326	67.033	>>>>> run number 20 is done!!! <<<<<
11.141	26.055		>>>>> run number 21 is done!!! <<<<<
11.166	25.961		>>>>> run number 22 is done!!! <<<<<
11.221	26.634	76.881	>>>>> run number 23 is done!!! <<<<<
11.356	26.55	130.907	>>>>> run number 24 is done!!! <<<<<
11.320	26.162	86.22	>>>>> run number 25 is done!!! <<<<<
11.172	25.926	55.277	>>>>> run number 26 is done!!! <<<<<
11.329	26.17	137.911	>>>>> run number 27 is done!!! <<<<<
11.176	26.512	109.304	>>>>> run number 28 is done!!! <<<<<
11.122	26.038	78.066	>>>>> run number 29 is done!!! <<<<<
11.140	26.236	131.858	>>>>> run number 30 is done!!! <<<<<
11.234	26.54	82.704	>>>>> run number 31 is done!!! <<<<<
11.422	26.044	87.545	>>>>> run number 32 is done!!! <<<<<
11.349	26.567		>>>>> run number 33 is done!!! <<<<<
11.188	26.113	143.141	>>>>> run number 34 is done!!! <<<<<
11.198	26.553	71.483	>>>>> run number 35 is done!!! <<<<<
11.171	26.459		>>>>> run number 36 is done!!! <<<<<
11.242	26.222	40.728	>>>>> run number 37 is done!!! <<<<<
11.247	26.327	58.858	>>>>> run number 38 is done!!! <<<<<
11.378	26.55	82.728	>>>>> run number 39 is done!!! <<<<<
11.174	26.542	64.761	>>>>> run number 40 is done!!! <<<<<
11.138	25.992	146.138	>>>>> run number 41 is done!!! <<<<<
11.140	26.554		>>>>> run number 42 is done!!! <<<<<
11.197	26.568		>>>>> run number 43 is done!!! <<<<<
11.205	26.624		>>>>> run number 44 is done!!! <<<<<
11.229	26.583	135.333	>>>>> run number 45 is done!!! <<<<<
11.198	26.559	88.545	>>>>> run number 46 is done!!! <<<<<
11.385	26.23	58.964	>>>>> run number 47 is done!!! <<<<<
11.509	26.55		>>>>> run number 48 is done!!! <<<<<
11.170	26.575	85.327	>>>>> run number 49 is done!!! <<<<<
11.193	26.362	82.056	>>>>> run number 50 is done!!! <<<<<
11.173	26.306	85.633	>>>>> run number 51 is done!!! <<<<<
11.195	26.568	41.337	>>>>> run number 52 is done!!! <<<<<
11.327	26.568		>>>>> run number 53 is done!!! <<<<<
11.329	26.032	84.616	>>>>> run number 54 is done!!! <<<<<
11.258	26.549	144.616	>>>>> run number 55 is done!!! <<<<<
11.269	26.008	39.906	>>>>> run number 56 is done!!! <<<<<
11.335	26.117	78.312	>>>>> run number 57 is done!!! <<<<<

11.310	26.629	61.086	>>>>> run number 58 is done!!! <<<<<
11.292	26.266	129.928	>>>>> run number 59 is done!!! <<<<<
11.188	26.315		>>>>> run number 60 is done!!! <<<<<
11.174	26.142	82.053	>>>>> run number 61 is done!!! <<<<<
11.341	26.558	74.019	>>>>> run number 62 is done!!! <<<<<
11.301	26.144		>>>>> run number 63 is done!!! <<<<<
11.435	26.114	66.879	>>>>> run number 64 is done!!! <<<<<
11.287	26.236	131.412	>>>>> run number 65 is done!!! <<<<<
11.148	26.575	83.733	>>>>> run number 66 is done!!! <<<<<
11.202	26.565		>>>>> run number 67 is done!!! <<<<<
11.275	26.294		>>>>> run number 68 is done!!! <<<<<
11.218	26.163	64.1	>>>>> run number 69 is done!!! <<<<<
11.518	26.335		>>>>> run number 70 is done!!! <<<<<
11.214	26.185		>>>>> run number 71 is done!!! <<<<<
11.178	26.116		>>>>> run number 72 is done!!! <<<<<
11.158	26.596		>>>>> run number 73 is done!!! <<<<<
11.167	26.363		>>>>> run number 74 is done!!! <<<<<
11.133	26.557	93.028	>>>>> run number 75 is done!!! <<<<<
11.464	26.14	127.735	>>>>> run number 76 is done!!! <<<<<
11.313	26.228	79.428	>>>>> run number 77 is done!!! <<<<<
11.188	26.224		>>>>> run number 78 is done!!! <<<<<
11.382	26.584	41.407	>>>>> run number 79 is done!!! <<<<<
11.281	26.579	128.732	>>>>> run number 80 is done!!! <<<<<
11.139	26.55		>>>>> run number 81 is done!!! <<<<<
11.366	26.447		>>>>> run number 82 is done!!! <<<<<
11.395	25.907	83.527	>>>>> run number 83 is done!!! <<<<<
11.230	26.448	47.424	>>>>> run number 84 is done!!! <<<<<
11.182	25.983		>>>>> run number 85 is done!!! <<<<<
11.347	26.46		>>>>> run number 86 is done!!! <<<<<
11.298	26.465	96.928	>>>>> run number 87 is done!!! <<<<<
11.255	26.227	144.021	>>>>> run number 88 is done!!! <<<<<
11.162	25.974	62.498	>>>>> run number 89 is done!!! <<<<<
11.431	25.907	73.102	>>>>> run number 90 is done!!! <<<<<
11.363	26.362		>>>>> run number 91 is done!!! <<<<<
11.152	26.591	85.415	>>>>> run number 92 is done!!! <<<<<
11.330	26.576		>>>>> run number 93 is done!!! <<<<<
11.582	26.575	130.318	>>>>> run number 94 is done!!! <<<<<
11.245	26.171		>>>>> run number 95 is done!!! <<<<<
11.288	26.116	81.668	>>>>> run number 96 is done!!! <<<<<
11.320	26.291		>>>>> run number 97 is done!!! <<<<<
11.121	25.893	98.933	>>>>> run number 98 is done!!! <<<<<
11.378	26.046		>>>>> run number 99 is done!!! <<<<<
11.168	26.558	69.61	>>>>> run number 100 is done!!! <<<<<

inputFileName[i] = ../scenarios/analysis_ff_05			
fD	fS	fK	
16.683	57.584	63.888	>>>>> run number 1 is done!!! <<<<<
16.676	57.619	62.704	>>>>> run number 2 is done!!! <<<<<
16.766	58.636	62.916	>>>>> run number 3 is done!!! <<<<<
17.104	58.101	68.913	>>>>> run number 4 is done!!! <<<<<
16.706	57.886	61.273	>>>>> run number 5 is done!!! <<<<<
16.817	57.506	63.7	>>>>> run number 6 is done!!! <<<<<

16.844	57.04	60.316	>>>>> run number 7 is done!!! <<<<<
17.134	56.537	58.727	>>>>> run number 8 is done!!! <<<<<
16.735	52.83	65.025	>>>>> run number 9 is done!!! <<<<<
16.999	57.315	65.472	>>>>> run number 10 is done!!! <<<<<
16.832	60.526	65.613	>>>>> run number 11 is done!!! <<<<<
16.929	57.678	68.123	>>>>> run number 12 is done!!! <<<<<
16.677	56.003	72.085	>>>>> run number 13 is done!!! <<<<<
17.114	53.744	67.001	>>>>> run number 14 is done!!! <<<<<
16.725	57.166	60.782	>>>>> run number 15 is done!!! <<<<<
17.092	58.152	64.588	>>>>> run number 16 is done!!! <<<<<
16.919	57.541	64.453	>>>>> run number 17 is done!!! <<<<<
16.716	57.441	59.107	>>>>> run number 18 is done!!! <<<<<
16.694	52.277	59.38	>>>>> run number 19 is done!!! <<<<<
17.129	57.155	63.779	>>>>> run number 20 is done!!! <<<<<
17.024	60.737	68.313	>>>>> run number 21 is done!!! <<<<<
16.703	57.439	62.635	>>>>> run number 22 is done!!! <<<<<
16.996	58.771	60.666	>>>>> run number 23 is done!!! <<<<<
17.033	61.183	62.738	>>>>> run number 24 is done!!! <<<<<
17.060	57.079	64.116	>>>>> run number 25 is done!!! <<<<<
16.746	60.984	66.516	>>>>> run number 26 is done!!! <<<<<
16.997	60.738	64.037	>>>>> run number 27 is done!!! <<<<<
16.928	60.767	63.179	>>>>> run number 28 is done!!! <<<<<
16.929	57.6	61.547	>>>>> run number 29 is done!!! <<<<<
16.809	57.581	60.934	>>>>> run number 30 is done!!! <<<<<
16.689	57.81	68.867	>>>>> run number 31 is done!!! <<<<<
16.824	60.735	62.074	>>>>> run number 32 is done!!! <<<<<
16.814	57.945	68.267	>>>>> run number 33 is done!!! <<<<<
16.861	57.839	63.702	>>>>> run number 34 is done!!! <<<<<
16.997	57.799	62.205	>>>>> run number 35 is done!!! <<<<<
16.800	62.341	63.705	>>>>> run number 36 is done!!! <<<<<
17.004	57.341	59.953	>>>>> run number 37 is done!!! <<<<<
16.989	57.577	59.963	>>>>> run number 38 is done!!! <<<<<
17.021	57.76	65.54	>>>>> run number 39 is done!!! <<<<<
16.714	60.777	73.869	>>>>> run number 40 is done!!! <<<<<
16.853	56.98	61.348	>>>>> run number 41 is done!!! <<<<<
17.060	60.817	62.856	>>>>> run number 42 is done!!! <<<<<
17.150	56.54	62.52	>>>>> run number 43 is done!!! <<<<<
16.738	57.713	65.925	>>>>> run number 44 is done!!! <<<<<
17.083	57.4	64.559	>>>>> run number 45 is done!!! <<<<<
16.897	60.5	71.614	>>>>> run number 46 is done!!! <<<<<
17.073	58.556	60.347	>>>>> run number 47 is done!!! <<<<<
17.081	57.928	70.296	>>>>> run number 48 is done!!! <<<<<
16.720	57.302	62.8	>>>>> run number 49 is done!!! <<<<<
17.056	57.97	62.767	>>>>> run number 50 is done!!! <<<<<
16.703	58.413	60.275	>>>>> run number 51 is done!!! <<<<<
17.008	58.714	63.552	>>>>> run number 52 is done!!! <<<<<
16.768	55.549	73.944	>>>>> run number 53 is done!!! <<<<<
16.770	57.628	68.17	>>>>> run number 54 is done!!! <<<<<
17.106	60.487	65.28	>>>>> run number 55 is done!!! <<<<<
16.869	57.684	65.081	>>>>> run number 56 is done!!! <<<<<
16.992	57.979	68.689	>>>>> run number 57 is done!!! <<<<<
17.036	57.769	64.693	>>>>> run number 58 is done!!! <<<<<
17.043	57.473	73.777	>>>>> run number 59 is done!!! <<<<<
16.738	56.831	61.37	>>>>> run number 60 is done!!! <<<<<

17.089	57.305	58.865	>>>>> run number 61 is done!!! <<<<<
16.663	57.365	60.769	>>>>> run number 62 is done!!! <<<<<
17.020	57.479	59.56	>>>>> run number 63 is done!!! <<<<<
16.712	57.828	59.401	>>>>> run number 64 is done!!! <<<<<
16.688	60.899	68.27	>>>>> run number 65 is done!!! <<<<<
16.709	58.085	68.571	>>>>> run number 66 is done!!! <<<<<
16.883	58.233	67.985	>>>>> run number 67 is done!!! <<<<<
16.808	57.439	62.186	>>>>> run number 68 is done!!! <<<<<
16.719	60.42	67.507	>>>>> run number 69 is done!!! <<<<<
16.697	57.178	60.031	>>>>> run number 70 is done!!! <<<<<
16.661	57.611	59.602	>>>>> run number 71 is done!!! <<<<<
16.773	61.148	71.806	>>>>> run number 72 is done!!! <<<<<
16.770	57.724	62.523	>>>>> run number 73 is done!!! <<<<<
17.001	52.572	62.019	>>>>> run number 74 is done!!! <<<<<
17.012	58.164	62.793	>>>>> run number 75 is done!!! <<<<<
16.831	57.324	66.297	>>>>> run number 76 is done!!! <<<<<
16.665	57.165	58.457	>>>>> run number 77 is done!!! <<<<<
16.734	56.914	63.989	>>>>> run number 78 is done!!! <<<<<
17.053	60.848	69.524	>>>>> run number 79 is done!!! <<<<<
16.724	57.288	63.525	>>>>> run number 80 is done!!! <<<<<
16.961	56.791	69.52	>>>>> run number 81 is done!!! <<<<<
16.774	57.553	63.993	>>>>> run number 82 is done!!! <<<<<
17.015	56.965	63.386	>>>>> run number 83 is done!!! <<<<<
16.829	57.75	62.945	>>>>> run number 84 is done!!! <<<<<
16.732	57.76	59.824	>>>>> run number 85 is done!!! <<<<<
16.752	57.771	64.561	>>>>> run number 86 is done!!! <<<<<
17.077	58.003	65.3	>>>>> run number 87 is done!!! <<<<<
17.125	56.949	59.958	>>>>> run number 88 is done!!! <<<<<
16.753	56.89	60.733	>>>>> run number 89 is done!!! <<<<<
16.765	57.286	65.609	>>>>> run number 90 is done!!! <<<<<
17.001	58.893	60.681	>>>>> run number 91 is done!!! <<<<<
17.013	57.804	67.304	>>>>> run number 92 is done!!! <<<<<
17.217	61.387	65.7	>>>>> run number 93 is done!!! <<<<<
16.704	58.174	66.509	>>>>> run number 94 is done!!! <<<<<
16.775	57.495	62.109	>>>>> run number 95 is done!!! <<<<<
16.742	57.228	70.586	>>>>> run number 96 is done!!! <<<<<
16.784	57.528	62.878	>>>>> run number 97 is done!!! <<<<<
16.909	57.615	61.463	>>>>> run number 98 is done!!! <<<<<
17.139	55.667	63.767	>>>>> run number 99 is done!!! <<<<<
16.990	57.17	64.411	>>>>> run number 100 is done!!! <<<<<

inputFileName[i] = ../scenarios/analysis_ff_06			
fD			
19.573			>>>>> run number 1 is done!!! <<<<<
19.504			>>>>> run number 2 is done!!! <<<<<
17.665			>>>>> run number 3 is done!!! <<<<<
19.536			>>>>> run number 4 is done!!! <<<<<
19.381			>>>>> run number 5 is done!!! <<<<<
19.383			>>>>> run number 6 is done!!! <<<<<
17.636			>>>>> run number 7 is done!!! <<<<<
19.511			>>>>> run number 8 is done!!! <<<<<
19.442			>>>>> run number 9 is done!!! <<<<<

19.423			>>>>> run number 10 is done!!! <<<<<
19.369			>>>>> run number 11 is done!!! <<<<<
19.378			>>>>> run number 12 is done!!! <<<<<
19.397			>>>>> run number 13 is done!!! <<<<<
17.647			>>>>> run number 14 is done!!! <<<<<
19.371			>>>>> run number 15 is done!!! <<<<<
19.367			>>>>> run number 16 is done!!! <<<<<
19.465			>>>>> run number 17 is done!!! <<<<<
19.423			>>>>> run number 18 is done!!! <<<<<
19.409			>>>>> run number 19 is done!!! <<<<<
19.415			>>>>> run number 20 is done!!! <<<<<
19.510			>>>>> run number 21 is done!!! <<<<<
19.403			>>>>> run number 22 is done!!! <<<<<
19.500			>>>>> run number 23 is done!!! <<<<<
19.413			>>>>> run number 24 is done!!! <<<<<
19.493			>>>>> run number 25 is done!!! <<<<<
19.406			>>>>> run number 26 is done!!! <<<<<
19.406			>>>>> run number 27 is done!!! <<<<<
19.401			>>>>> run number 28 is done!!! <<<<<
19.479			>>>>> run number 29 is done!!! <<<<<
19.454			>>>>> run number 30 is done!!! <<<<<
19.426			>>>>> run number 31 is done!!! <<<<<
19.390			>>>>> run number 32 is done!!! <<<<<
19.427			>>>>> run number 33 is done!!! <<<<<
19.387			>>>>> run number 34 is done!!! <<<<<
19.539			>>>>> run number 35 is done!!! <<<<<
19.503			>>>>> run number 36 is done!!! <<<<<
19.389			>>>>> run number 37 is done!!! <<<<<
17.561			>>>>> run number 38 is done!!! <<<<<
19.405			>>>>> run number 39 is done!!! <<<<<
19.430			>>>>> run number 40 is done!!! <<<<<
19.450			>>>>> run number 41 is done!!! <<<<<
19.402			>>>>> run number 42 is done!!! <<<<<
19.504			>>>>> run number 43 is done!!! <<<<<
19.419			>>>>> run number 44 is done!!! <<<<<
19.385			>>>>> run number 45 is done!!! <<<<<
19.461			>>>>> run number 46 is done!!! <<<<<
19.369			>>>>> run number 47 is done!!! <<<<<
19.404			>>>>> run number 48 is done!!! <<<<<
19.416			>>>>> run number 49 is done!!! <<<<<
19.515			>>>>> run number 50 is done!!! <<<<<
19.385			>>>>> run number 51 is done!!! <<<<<
19.413			>>>>> run number 52 is done!!! <<<<<
19.387			>>>>> run number 53 is done!!! <<<<<
19.398			>>>>> run number 54 is done!!! <<<<<
19.377			>>>>> run number 55 is done!!! <<<<<
17.598			>>>>> run number 56 is done!!! <<<<<
19.560			>>>>> run number 57 is done!!! <<<<<
17.589			>>>>> run number 58 is done!!! <<<<<
19.523			>>>>> run number 59 is done!!! <<<<<
19.394			>>>>> run number 60 is done!!! <<<<<
19.607			>>>>> run number 61 is done!!! <<<<<
19.400			>>>>> run number 62 is done!!! <<<<<
19.491			>>>>> run number 63 is done!!! <<<<<

19.397			>>>>> run number 64 is done!!! <<<<<
19.383			>>>>> run number 65 is done!!! <<<<<
19.465			>>>>> run number 66 is done!!! <<<<<
17.647			>>>>> run number 67 is done!!! <<<<<
19.421			>>>>> run number 68 is done!!! <<<<<
19.427			>>>>> run number 69 is done!!! <<<<<
19.386			>>>>> run number 70 is done!!! <<<<<
19.455			>>>>> run number 71 is done!!! <<<<<
19.388			>>>>> run number 72 is done!!! <<<<<
19.401			>>>>> run number 73 is done!!! <<<<<
19.420			>>>>> run number 74 is done!!! <<<<<
19.445			>>>>> run number 75 is done!!! <<<<<
17.542			>>>>> run number 76 is done!!! <<<<<
19.390			>>>>> run number 77 is done!!! <<<<<
19.402			>>>>> run number 78 is done!!! <<<<<
19.527			>>>>> run number 79 is done!!! <<<<<
19.487			>>>>> run number 80 is done!!! <<<<<
19.374			>>>>> run number 81 is done!!! <<<<<
19.424			>>>>> run number 82 is done!!! <<<<<
19.381			>>>>> run number 83 is done!!! <<<<<
19.485			>>>>> run number 84 is done!!! <<<<<
19.456			>>>>> run number 85 is done!!! <<<<<
17.582			>>>>> run number 86 is done!!! <<<<<
19.793			>>>>> run number 87 is done!!! <<<<<
19.410			>>>>> run number 88 is done!!! <<<<<
19.375			>>>>> run number 89 is done!!! <<<<<
19.379			>>>>> run number 90 is done!!! <<<<<
19.413			>>>>> run number 91 is done!!! <<<<<
19.418			>>>>> run number 92 is done!!! <<<<<
19.454			>>>>> run number 93 is done!!! <<<<<
19.374			>>>>> run number 94 is done!!! <<<<<
19.378			>>>>> run number 95 is done!!! <<<<<
19.512			>>>>> run number 96 is done!!! <<<<<
19.364			>>>>> run number 97 is done!!! <<<<<
19.406			>>>>> run number 98 is done!!! <<<<<
19.398			>>>>> run number 99 is done!!! <<<<<
19.392			>>>>> run number 100 is done!!! <<<<<

inputFileName[i] = ../scenarios/analysis_ff_07			
fD	fS	fK	
9.915	42.743	81.43	>>>>> run number 1 is done!!! <<<<<
10.288	42.351	42.351	>>>>> run number 2 is done!!! <<<<<
11.704	43.141	71.107	>>>>> run number 3 is done!!! <<<<<
10.755	42.742	80.299	>>>>> run number 4 is done!!! <<<<<
11.995	43.024	43.024	>>>>> run number 5 is done!!! <<<<<
12.933	42.913	75.406	>>>>> run number 6 is done!!! <<<<<
9.863	67.98	83.784	>>>>> run number 7 is done!!! <<<<<
10.719	24.441	83.401	>>>>> run number 8 is done!!! <<<<<
10.327	42.633	76.256	>>>>> run number 9 is done!!! <<<<<
13.156	42.849	80.611	>>>>> run number 10 is done!!! <<<<<
11.897	24.183	76.047	>>>>> run number 11 is done!!! <<<<<
12.733	24.465	70.462	>>>>> run number 12 is done!!! <<<<<

11.581	61.848	75.126	>>>>> run number 13 is done!!! <<<<<
10.463	63.08	77.471	>>>>> run number 14 is done!!! <<<<<
11.024	42.302	86.574	>>>>> run number 15 is done!!! <<<<<
10.610	24.408	77.229	>>>>> run number 16 is done!!! <<<<<
11.004	24.479	78.25	>>>>> run number 17 is done!!! <<<<<
9.590	42.595	74.759	>>>>> run number 18 is done!!! <<<<<
12.188	42.787	64.658	>>>>> run number 19 is done!!! <<<<<
9.897	42.39	65.49	>>>>> run number 20 is done!!! <<<<<
11.748	42.809	78.814	>>>>> run number 21 is done!!! <<<<<
10.732	42.688	42.688	>>>>> run number 22 is done!!! <<<<<
9.880	64.343	88.1	>>>>> run number 23 is done!!! <<<<<
10.189	62.43	65.243	>>>>> run number 24 is done!!! <<<<<
11.525	42.66	77.904	>>>>> run number 25 is done!!! <<<<<
11.141	60.838	77.307	>>>>> run number 26 is done!!! <<<<<
10.013	42.329	83.014	>>>>> run number 27 is done!!! <<<<<
10.849	24.34	74.382	>>>>> run number 28 is done!!! <<<<<
10.829	63.116	97.092	>>>>> run number 29 is done!!! <<<<<
10.550	63.771	67.055	>>>>> run number 30 is done!!! <<<<<
9.684	42.888	66.382	>>>>> run number 31 is done!!! <<<<<
12.026	42.631	75.941	>>>>> run number 32 is done!!! <<<<<
12.325	42.467	74.789	>>>>> run number 33 is done!!! <<<<<
11.200	42.479	75.906	>>>>> run number 34 is done!!! <<<<<
11.940	42.514	76.858	>>>>> run number 35 is done!!! <<<<<
12.298	42.659	78.018	>>>>> run number 36 is done!!! <<<<<
10.934	42.715	62.329	>>>>> run number 37 is done!!! <<<<<
10.994	24.627	76.091	>>>>> run number 38 is done!!! <<<<<
11.992	24.559	77.807	>>>>> run number 39 is done!!! <<<<<
12.074	42.532	83.433	>>>>> run number 40 is done!!! <<<<<
10.006	64.959	76.548	>>>>> run number 41 is done!!! <<<<<
11.877	24.536	70.263	>>>>> run number 42 is done!!! <<<<<
11.871	24.527	75.715	>>>>> run number 43 is done!!! <<<<<
11.696	42.614	80.334	>>>>> run number 44 is done!!! <<<<<
12.347	42.989	72.554	>>>>> run number 45 is done!!! <<<<<
12.085	42.823	75.939	>>>>> run number 46 is done!!! <<<<<
10.431	63.792	64.508	>>>>> run number 47 is done!!! <<<<<
9.545	64.463	66.065	>>>>> run number 48 is done!!! <<<<<
13.457	43.043	74.739	>>>>> run number 49 is done!!! <<<<<
11.458	42.605	42.605	>>>>> run number 50 is done!!! <<<<<
12.534	42.426	73.426	>>>>> run number 51 is done!!! <<<<<
11.367	24.387	74.823	>>>>> run number 52 is done!!! <<<<<
10.397	42.522	78.076	>>>>> run number 53 is done!!! <<<<<
10.165	62.709	78.721	>>>>> run number 54 is done!!! <<<<<
12.971	24.699	65.769	>>>>> run number 55 is done!!! <<<<<
12.273	24.488	75.774	>>>>> run number 56 is done!!! <<<<<
10.021	61.434	95.706	>>>>> run number 57 is done!!! <<<<<
9.571	42.501	74.588	>>>>> run number 58 is done!!! <<<<<
12.420	66.778	74.173	>>>>> run number 59 is done!!! <<<<<
11.495	42.79	63.614	>>>>> run number 60 is done!!! <<<<<
10.826	24.494	101.234	>>>>> run number 61 is done!!! <<<<<
10.596	24.459	74.55	>>>>> run number 62 is done!!! <<<<<
12.606	65.817	79.673	>>>>> run number 63 is done!!! <<<<<
9.921	42.764	78.559	>>>>> run number 64 is done!!! <<<<<
11.603	43.005	65.123	>>>>> run number 65 is done!!! <<<<<
12.136	43.094	74.691	>>>>> run number 66 is done!!! <<<<<

11.509	24.14	67.54	>>>>> run number 67 is done!!! <<<<<
10.766	42.883	76.458	>>>>> run number 68 is done!!! <<<<<
10.432	42.526	73.921	>>>>> run number 69 is done!!! <<<<<
11.899	42.672	66.769	>>>>> run number 70 is done!!! <<<<<
9.517	61.842	81.308	>>>>> run number 71 is done!!! <<<<<
10.825	62.981	77.559	>>>>> run number 72 is done!!! <<<<<
11.775	62.678	72.325	>>>>> run number 73 is done!!! <<<<<
12.559	42.982	62.041	>>>>> run number 74 is done!!! <<<<<
10.598	63.321	83.795	>>>>> run number 75 is done!!! <<<<<
10.433	24.484	81.51	>>>>> run number 76 is done!!! <<<<<
10.042	42.806	74.585	>>>>> run number 77 is done!!! <<<<<
10.854	42.656	72.583	>>>>> run number 78 is done!!! <<<<<
11.239	62.582	76.662	>>>>> run number 79 is done!!! <<<<<
10.135	42.761	76.993	>>>>> run number 80 is done!!! <<<<<
10.883	62.534	82.029	>>>>> run number 81 is done!!! <<<<<
10.851	42.554	61.757	>>>>> run number 82 is done!!! <<<<<
10.389	42.552	60.975	>>>>> run number 83 is done!!! <<<<<
10.760	42.651	76.391	>>>>> run number 84 is done!!! <<<<<
10.298	24.286	75.972	>>>>> run number 85 is done!!! <<<<<
10.734	24.386	71.426	>>>>> run number 86 is done!!! <<<<<
12.554	67.206	72.479	>>>>> run number 87 is done!!! <<<<<
11.702	42.808	61.892	>>>>> run number 88 is done!!! <<<<<
9.816	64.994	100.453	>>>>> run number 89 is done!!! <<<<<
11.477	62.439	78.44	>>>>> run number 90 is done!!! <<<<<
12.052	24.419	74.576	>>>>> run number 91 is done!!! <<<<<
10.096	66.219	79.022	>>>>> run number 92 is done!!! <<<<<
13.058	65.336	69.188	>>>>> run number 93 is done!!! <<<<<
11.388	24.489	76.228	>>>>> run number 94 is done!!! <<<<<
10.895	42.671	76.44	>>>>> run number 95 is done!!! <<<<<
9.709	42.654	79.429	>>>>> run number 96 is done!!! <<<<<
9.674	42.721	42.721	>>>>> run number 97 is done!!! <<<<<
12.807	24.575	42.537	>>>>> run number 98 is done!!! <<<<<
10.385	55.356	66.709	>>>>> run number 99 is done!!! <<<<<
11.040	63.754	74.267	>>>>> run number 100 is done!!! <<<<<

inputFileName[i] = ../scenarios/analysis_ff_08			
fD	fS	fK	
20.615	59.706	69.152	>>>>> run number 1 is done!!! <<<<<
20.940	60.291	70.478	>>>>> run number 2 is done!!! <<<<<
21.002	60.726	78.814	>>>>> run number 3 is done!!! <<<<<
20.671			>>>>> run number 4 is done!!! <<<<<
21.206			>>>>> run number 5 is done!!! <<<<<
21.077			>>>>> run number 6 is done!!! <<<<<
20.607	60.355	75.89	>>>>> run number 7 is done!!! <<<<<
20.645			>>>>> run number 8 is done!!! <<<<<
21.154	60.834	72.936	>>>>> run number 9 is done!!! <<<<<
20.962			>>>>> run number 10 is done!!! <<<<<
21.218			>>>>> run number 11 is done!!! <<<<<
21.367	62.029	67.607	>>>>> run number 12 is done!!! <<<<<
20.830	60.079	73.239	>>>>> run number 13 is done!!! <<<<<
21.172			>>>>> run number 14 is done!!! <<<<<
21.127			>>>>> run number 15 is done!!! <<<<<

21.079			>>>>> run number 16 is done!!! <<<<<
21.178	60.836	73.884	>>>>> run number 17 is done!!! <<<<<
21.401	62.093	71.022	>>>>> run number 18 is done!!! <<<<<
21.065			>>>>> run number 19 is done!!! <<<<<
21.043	60.521	71.346	>>>>> run number 20 is done!!! <<<<<
21.292			>>>>> run number 21 is done!!! <<<<<
21.280			>>>>> run number 22 is done!!! <<<<<
21.182			>>>>> run number 23 is done!!! <<<<<
21.344	61.812	70.282	>>>>> run number 24 is done!!! <<<<<
21.019			>>>>> run number 25 is done!!! <<<<<
21.160	60.591	73.44	>>>>> run number 26 is done!!! <<<<<
21.410			>>>>> run number 27 is done!!! <<<<<
21.352	62.047	71.837	>>>>> run number 28 is done!!! <<<<<
21.200	60.606	71.475	>>>>> run number 29 is done!!! <<<<<
21.378	61.413	77.461	>>>>> run number 30 is done!!! <<<<<
21.299			>>>>> run number 31 is done!!! <<<<<
21.414	61.874	73.498	>>>>> run number 32 is done!!! <<<<<
21.051	49.952	80.462	>>>>> run number 33 is done!!! <<<<<
21.081	60.999	64.283	>>>>> run number 34 is done!!! <<<<<
21.351	61.451	74.278	>>>>> run number 35 is done!!! <<<<<
21.270			>>>>> run number 36 is done!!! <<<<<
20.578	60.188	70.706	>>>>> run number 37 is done!!! <<<<<
20.948			>>>>> run number 38 is done!!! <<<<<
20.740			>>>>> run number 39 is done!!! <<<<<
20.866			>>>>> run number 40 is done!!! <<<<<
20.608	59.94	72.303	>>>>> run number 41 is done!!! <<<<<
21.143			>>>>> run number 42 is done!!! <<<<<
21.243			>>>>> run number 43 is done!!! <<<<<
21.381	62.068	72.969	>>>>> run number 44 is done!!! <<<<<
20.992			>>>>> run number 45 is done!!! <<<<<
21.264	61.306	78.568	>>>>> run number 46 is done!!! <<<<<
21.343	61.996	73.531	>>>>> run number 47 is done!!! <<<<<
21.162			>>>>> run number 48 is done!!! <<<<<
20.991			>>>>> run number 49 is done!!! <<<<<
21.371	62.055	72.784	>>>>> run number 50 is done!!! <<<<<
21.376	62.036	71.022	>>>>> run number 51 is done!!! <<<<<
21.152	60.944	71.18	>>>>> run number 52 is done!!! <<<<<
20.806			>>>>> run number 53 is done!!! <<<<<
20.869	60.744	71.497	>>>>> run number 54 is done!!! <<<<<
21.320			>>>>> run number 55 is done!!! <<<<<
20.727	60.409	70.569	>>>>> run number 56 is done!!! <<<<<
21.076	61.077	71.153	>>>>> run number 57 is done!!! <<<<<
20.852			>>>>> run number 58 is done!!! <<<<<
20.827			>>>>> run number 59 is done!!! <<<<<
20.959			>>>>> run number 60 is done!!! <<<<<
21.075			>>>>> run number 61 is done!!! <<<<<
21.181	61.212	71.141	>>>>> run number 62 is done!!! <<<<<
20.845			>>>>> run number 63 is done!!! <<<<<
21.377			>>>>> run number 64 is done!!! <<<<<
21.157	60.738	71.707	>>>>> run number 65 is done!!! <<<<<
21.454			>>>>> run number 66 is done!!! <<<<<
21.355	61.704	72.217	>>>>> run number 67 is done!!! <<<<<
21.205	61.253	64.476	>>>>> run number 68 is done!!! <<<<<
20.734			>>>>> run number 69 is done!!! <<<<<

21.184	61.244	78.017	>>>>> run number 70 is done!!! <<<<<
21.369			>>>>> run number 71 is done!!! <<<<<
21.112			>>>>> run number 72 is done!!! <<<<<
20.908	60.773	76.097	>>>>> run number 73 is done!!! <<<<<
21.327			>>>>> run number 74 is done!!! <<<<<
21.254			>>>>> run number 75 is done!!! <<<<<
21.054			>>>>> run number 76 is done!!! <<<<<
20.667	60.434	70.872	>>>>> run number 77 is done!!! <<<<<
21.129	61.01	69.042	>>>>> run number 78 is done!!! <<<<<
21.137			>>>>> run number 79 is done!!! <<<<<
21.149	61.114	74.201	>>>>> run number 80 is done!!! <<<<<
21.248			>>>>> run number 81 is done!!! <<<<<
21.165			>>>>> run number 82 is done!!! <<<<<
21.390			>>>>> run number 83 is done!!! <<<<<
21.222			>>>>> run number 84 is done!!! <<<<<
21.435			>>>>> run number 85 is done!!! <<<<<
20.641			>>>>> run number 86 is done!!! <<<<<
20.746	60.029	71.692	>>>>> run number 87 is done!!! <<<<<
21.280			>>>>> run number 88 is done!!! <<<<<
21.391	61.848	74.456	>>>>> run number 89 is done!!! <<<<<
20.732			>>>>> run number 90 is done!!! <<<<<
21.345			>>>>> run number 91 is done!!! <<<<<
21.257			>>>>> run number 92 is done!!! <<<<<
20.893	60.813	79.769	>>>>> run number 93 is done!!! <<<<<
21.367			>>>>> run number 94 is done!!! <<<<<
20.561			>>>>> run number 95 is done!!! <<<<<
21.401	62.023	64.501	>>>>> run number 96 is done!!! <<<<<
20.605	60.155	85.747	>>>>> run number 97 is done!!! <<<<<
21.186			>>>>> run number 98 is done!!! <<<<<
21.159			>>>>> run number 99 is done!!! <<<<<
21.376	61.961	68.462	>>>>> run number 100 is done!!! <<<<<

APPENDIX C. JAVA CODE

The source code of the simulation model is free for all to use. The simulation model consists of 50 Java classes, out of which the longest one has almost 600 lines of code. Therefore, it seems useless to provide a printout of the source code. Further information, downloads, and up to date information may be found at http://diana.cs.nps.navy.mil/~wlehmann.

Any comments, recommendations, complaints, etc. are highly appreciated. Feel free to use my email address: wolfganglehmann@hotmail.com.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Bracken, J., Kress, M., and Rosenthal, R.E., *Warfare Modeling*, John Wiley & Sons, New York, 1996.

Brown, L.P., "Agent Based Simulation as an Exploratory Tool in the Study of Human Dimension," Masters Thesis, Naval Postgraduate School, Monterey, CA, March 2000.

Buss, A.H., "Modeling with Event Graphs," Proceedings of the 28th Conference on Winter Simulation, Coronado, CA, 1996.

Buss, A.H., "Basic Event Graph Modeling," Technical Notes, 2001, [http://diana.gl.nps.navy.mil/~ahbuss/papers/BasicEventGraphModeling.pdf].

Buss, A.H., "Discrete Event Programming with Simkit," Technical Notes, 2001, [http://diana.or.nps.navy.mil/~ahbuss/papers/Discrete%20Event%20Programming%20with%20Simkit.pdf, June 2006].

Buss, A.H., "Component Based Simulation Modeling with Simkit," Proceedings of the 34th conference on Winter Simulation, San Diego, CA, 2002.

Buss, A.H. and Sanchez, P.J., "Building Complex Models with LEGOS (Listener Event Graph Objects)," Proceedings of the 34th Conference on Winter Simulation, San Diego, CA, 2002.

Buss, A.H. and Sanchez, P.J., "Simple Movement and Detection in Discrete Event Simulation," Proceedings of the 2005 Winter Simulation Conference, Orlando, FL, 2005, [http://www.informs-cs.org/wsc05papers/118.pdf, June 2006].

Dewar, J.A., Gillogly, J.J., and Juncossa, M.L., "Non-Monotonicity, Chaos and Combat Models," Report R-3995-RC, Rand Corp., Santa Monica, CA, 1991.

Erlenbruch, T., "Agent-Based Simulation of German Peacekeeping Operations for Units up to Platoon Level," Masters Thesis, Naval Postgraduate School, Monterey, CA, March 2002.

Federal Minister of Defence, "Defence Policy Guidelines," Berlin 2003, [http://www.bmvg.de/portal/PA_1_0_LT/PortalFiles/C1256F1200608B1B/W268AHEH510INFOEN/VPR_en.pdf?yw_repository=youatweb, June 2006].

Federal Minister of Defence, "Bundeswehr Concept," Berlin 2004, [http://www.bmvg.de/portal/PA_1_0_LT/PortalFiles/C1256F1200608B1B/W268ADVU038INFOEN/KDB_en.pdf?yw_repository=youatweb, June 2006].

Gell-Mann, M., *The Quark and the Jaguar Adventures in the Simple and the Complex*, New York, W.J.Freemann and Company, New York, 1994.

Hamilton, L.C., *Regression with Graphics*, Wadsworth, Belmont, CA, 1992.

Holland, J., *Hidden Order*, Perseus Books, Cambridge, MA, 1995.

Horne, G. and Johnson, S., "Maneuver Warfare Science 2002," USMC Project Albert, Quantico, VA, 2002.

Hunter, D., *Beginning XML*, Wrox Press Ltd., Birmingham, UK, 2001.

Ilachinsky, A. "Irreducible Semi-Autonomous Adaptive Combat (ISAAC)," Center for Naval Analysis, Alexandria, VA, 1997.

Ipecki, A.I., "How Agent Based Models Can Be utilized to Explore and Exploit Non-Linearitiy and Intangibilities I Guerilla Warfare," Masters Thesis, Naval Postgraduate School, Monterey, CA, June 2002.

Kim, L., *The Official XMLSpy Handbook*, Wiley Publishing Inc., Indianapolis, IN, 2003.

Law, A.M. and Kelton, W.D., *Simulation Modeling and Analysis*, McGraw-Hill, Boston, MA, 2000.

Lorenz, P., Lecture Notes for the course "Simulation and Animation," March 2006, [http://isgwww.cs.uni-magdeburg.de/pelo/sa/sim1.php].

Lucas, T.W., Sanchez, S.M., Brown, L.P., and Vinyard, W.C., "Better Designs for High-Dimensional Explorations of Distillations," in [Horne, 2002].

Margolis, M., "Upgradeable Operational Availability Forecasting Tool for the U.S. Navy P 3 Replacement Aircraft," Masters Thesis, Naval Postgraduate School, Monterey, CA, 2003.

Nelson, J.J. et al., "Measures of Effectiveness for Humanitarian Assistance Operations," Center for Naval Analyses, Alexandria, VA; 1996.

Ray, E.T., *Learning XML*, O'Reilly & Associates Inc., Sebastopol, CA, 2001.

Savich, W., *Java. An Introduction to Computer Science and Programming*, Prentice Hall, Upper Saddle River, NJ, 2001.

Stolte, Maj M., "Modeling and Simulation in the German Army," 2001 [http://www.cdes.terre.defense.gouv.fr/sitefr/Objdoc/Numeros/fev_2001/ en/art13.pdf, March 2003].

Venables, W.N. and Ripley, B.D., *Modern Applied Statistics with S-PLUS*, Springer, New York, 2000.

Waldrop, M. M., *Complexity: the Emerging Science at the Edge of Order and Chaos*, Touchstone, New York, 1992.

Weiss, G., *Multiagent Systems*, MIT Press, Cambridge, MA, 1999.

Wellbrink, J., "Modeling Reduced Human Performance as a Complex Adaptive System," Ph.D. Dissertation, Naval Postgraduate School, Monterey, CA, September 2003.

Wooldridge, M., *An Introduction to Multiagent Systems*, John Wiley and Sons, Chinchester, West Sussex, 2002.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Fort Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Dr. Tom Lucas
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California

4. Dr. Arnold Buss
   MOVES Institute
   Naval Postgraduate School
   Monterey, California

5. Dr. Susan Sanchez
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California

6. OTL Dr. Wellbrink
   Heeresführungskommando
   Koblenz, Germany

7. Capt Michael Margolis
   Studies and Analysis Division, MCCDC, Code 19
   Quantico, Virginia